Information Processing Letters 112 (2012) 376–379

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On-line scheduling of equal-length intervals on parallel machines

Stanley P.Y. Fung [a], Chung Keung Poon [b,*], Duncan K.W. Yung [b]

[a] University of Leicester, Leicester, UK
[b] City University of Hong Kong, Hong Kong, China

ARTICLE INFO

Article history:
Received 22 September 2011
Received in revised form 2 January 2012
Accepted 31 January 2012
Available online 1 February 2012
Communicated by F.Y.L. Chin

Keywords:
Interval scheduling
Parallel scheduling
On-line algorithm
Competitive analysis

ABSTRACT

We consider the on-line preemptive scheduling of weighted equal-length intervals on multiple machines to maximize the total weight of completed intervals. We design an algorithm that is 2-competitive when the number of machines m is even; and $(2+\frac{2}{2m-1})$-competitive when m is an odd number at least 3. For example, when $m=3$, it is 2.4-competitive. As m increases, the competitive ratio approaches 2.

1. Introduction

We study the on-line preemptive scheduling of weighted intervals on m identical machines. The input consists of a set of intervals that arrive on-line and each machine is capable of processing one interval at a time. Our goal is to maximize the total weight of completed intervals. More precisely, each interval I has an *arrival time* $a(I)$, a *deadline* $d(I)$ and a *weight* $w(I)$. To complete I, the interval must be processed on a machine continuously from time $a(I)$ to $d(I)$ without interruption. We consider the on-line model, where the attributes of interval I are released at time $a(I)$ but not earlier. So, upon the arrival of I, the scheduler has to decide (immediately and without knowledge of the future) whether to start I and on which machine to process it, preempting the current interval there if any. Note that any interval that is not started immediately or is preempted before its completion will be lost forever.

Such a problem is motivated by many applications such as call control and bandwidth allocation (see [1,3,10]) in which jobs have fixed starting and finishing times and they arrive at their respective starting time, demanding immediate processing. When the system is overloaded, the scheduler has to decide whether to ignore a new job request or to preempt a currently running job in order to start the new one.

We gauge the performance of an on-line algorithm by its competitive ratio [9,2]. Given an input $\mathcal{I}$ (a set of intervals) and an algorithm A, we denote by $S_A(\mathcal{I})$ and $S^*(\mathcal{I})$ the schedules produced by A and by an optimal offline algorithm on $\mathcal{I}$, respectively. Denote by $|S|$ the total weight of intervals processed completely in S. Then the competitive ratio of algorithm A is defined as $r_A = \sup_{\mathcal{I}} \frac{|S^*(\mathcal{I})|}{|S_A(\mathcal{I})|}$.

Interval scheduling can be viewed as a special case of job scheduling in which the jobs always have tight deadlines, i.e., the processing time of a job is equal to the difference between its deadline and arrival time. Thus we define $d(I)-a(I)$ as the *processing time* (or *length*) of I. In this paper, we focus on equal-length intervals and assume w.l.o.g. that all intervals have length 1. Hence we have $d(I)=a(I)+1$.

We first mentioned the known results for weighted equal-length intervals. When there is only one machine ($m=1$), Woeginger [10] gave a deterministic 4-competitive algorithm (for more general intervals actually) and a matching lower bound. With randomization, the best upper and lower bounds are 2 (by Fung et al. [6]) and

* Corresponding author.
E-mail address: csckpoon@cityu.edu.hk (C.K. Poon).

$1+\ln(2)\approx 1.693$ (by Epstein and Levin [4]) respectively. The 2-competitive algorithm is in fact a *2-barely random* algorithm, i.e., one that randomly chooses between two deterministic algorithms at the very beginning and then sticks with the chosen algorithm thereafter. Fung et al. [6] gave a lower bound of 2 for the class of 2-barely random algorithms. For $m=2$, Fung et al. [7] presented a 3.582-competitive deterministic algorithm and gave a lower bound of 2. No results are known for $m\geqslant 3$.

For the on-line scheduling of unweighted variable-length intervals, Faigle and Nawijn [5] gave an optimal 1-competitive deterministic algorithm for m identical machines while Krumke et al. [8] studied the scheduling on related machines. These results are outside the scope of this paper.

In this paper, we design an on-line algorithm for our problem and prove that it is 2-competitive for all even m, improving the previous 3.582 upper bound when $m=2$ and generalizing it to larger m. The more interesting case is when m is odd and we design a novel scheme for distributing the intervals more evenly (w.r.t. their weights) among the machines. We show that our algorithm is $(2+\frac{2}{2m-1})$-competitive for all odd $m\geqslant 3$. Thus, the competitive ratio is 2.4 when $m=3$ and gradually approaches 2 as the number of machines m increases.

In the following section, we describe our algorithm for even m. In Section 3, we discuss the case of odd m and prove its competitive ratio. The paper is then concluded in Section 4 with some open problems.

2. Even number of machines

We first consider the case of two machines. Denote by A and B the two machines. We partition the time axis into slots $s_1, s_2, \ldots$ of unit length and assign A and B to take care of alternating slots as follows. For all odd slots s_i (i.e., i is odd), machine A will select the heaviest interval that arrives within slot s_i as follows. A begins by starting the first interval that arrives in s_i. When a new interval with heavier weight arrives within s_i, A preempts its current interval and starts the new one. Let I_i be the interval that A is processing when the end of slot s_i is reached. Clearly, I_i is the heaviest interval that arrives in slot s_i.

In the next slot s_{i+1}, A will run I_i to completion and then wait till the end of s_{i+1}, ignoring any new interval that arrives within s_{i+1}. At the same time, B will select the heaviest interval, I_{i+1}, that arrives in s_{i+1} using the same "greedy" method as A used in slot s_i. In the next slot s_{i+2}, B will complete I_{i+1} while A will select the heaviest interval that arrives in s_{i+2}, and so on. Since OPT can only serve the top-2 heaviest intervals arriving in each slot while our algorithm can serve at least the heaviest one, the competitive ratio is clearly 2.

The above idea can be easily generalized to all even m. Let $m=2q$ for some integer $q\geqslant 1$. We divide the machines into two groups, $\mathcal{A}$ and $\mathcal{B}$, each with q machines. In an odd slot s_i, machines in $\mathcal{A}$ select intervals that arrive within the slot using the same "greedy" method. That is, whenever a newly arrived interval has weight heavier than any interval currently being processed by machines in $\mathcal{A}$, the new interval will preempt the lightest one. In the following (even) slot s_{i+1}, machines in $\mathcal{A}$ complete their intervals while those in $\mathcal{B}$ select intervals that arrive in s_{i+1} greedily. Clearly, the algorithm can always obtain the weight of the top q heaviest intervals arriving in each slot while OPT can obtain at most the top $2q$ heaviest ones. Hence we have the following theorem:

Theorem 1. *The on-line algorithm has competitive ratio* 2 *when* m *is even.*

3. Odd number of machines

We first consider three machines. If, say, we allocate two machines for the odd slots and one machine for the even slots, then one can prove that the algorithm is 3-competitive using the same argument as in the previous section. This bound is also tight. Consider an input in which intervals only arrive in the even slots but not the odd slots. Then essentially the algorithm has to handle them with one machine while OPT can make use of all three machines. To get a smaller competitive ratio, one needs to have a way of sharing the 3 machines between two slots more evenly.

3.1. The algorithm

Let the three machines be A, B and C. In slot s_1, machine A and B will select intervals that arrive in the slot greedily as follows. They begin by starting the first two intervals that arrived in s_1. Whenever a new interval I arrives within slot s_1 and is of heavier weight than either of the two intervals currently being processed by A and B, the machine processing the lighter interval will preempt its interval and start I. Thus, at any moment A and B will be processing the top 2 heaviest intervals arriving in s_1 so far.

Let I_1 and J_1 be the intervals processed by A and B respectively when slot s_1 ends. Without loss of generality, assume $w(I_1)\geqslant w(J_1)$. In slot s_2, machine A will complete I_1 and then wait until the end of s_2. Machine B and C will take care of new intervals that arrive in slot s_2. The key idea of the algorithm is that B will only abort J_1 when it can start an interval of large enough weight. In more detail, the algorithm will go through one or more of the following states:

State 1: (*B has not completed J_1 while C is processing an interval of weight less than* $2w(J_1)$) In this state, C will select new intervals greedily. Note that at the beginning of slot s_2, C is idle and considered to be processing an interval of weight 0. Hence the algorithm is in State 1 initially. If C finally starts an interval of weight at least $2w(J_1)$, the algorithm goes to State 2. Otherwise if B has completed J_1, the algorithm goes to State 3.

State 2: (*B has not completed J_1 while C is processing an interval of weight at least* $2w(J_1)$) In this state, B and C will select new intervals with weight at least $2w(J_1)$ greedily. That is, if the new interval I has weight $w(I)<2w(J_1)$, we will ignore I. Otherwise, if $w(I)$ is larger than any of the current