Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 939717, 7 pages
doi:10.1155/2012/939717

Research Article

Online Parallel Machine Scheduling to Maximize the Number of Early Jobs

Feifeng Zheng,[1,2] Ming Liu,[3,4] Chengbin Chu,[3,4] and Yinfeng Xu[2]

[1] *Glorious Sun School of Business and Management, Donghua University, Shanghai 200092, China*
[2] *School of Management, Xi'an Jiaotong University, Xi'an, Shaanxi Province 710049, China*
[3] *School of Economics & Management, Tongji University, Shanghai 200092, China*
[4] *Laboratoire Génie Industriel, Ecole Centrale Paris, Grande Voie des Vignes, 92295 Châtenay-Malabry Cedex, France*

Correspondence should be addressed to Feifeng Zheng, zhengff@mail.xjtu.edu.cn

Received 4 August 2011; Accepted 17 November 2011

Academic Editor: Furong Gao

We study a maximization problem: online scheduling on m identical machines to maximize the number of early jobs. The problem is online in the sense that all jobs arrive over time. Each job's characteristics, such as processing time and due date, become known at its arrival time. We consider the *preemption-restart model*, in which preemption is allowed, while once a job is restarted, it loses all the progress that has been made on this job so far. If in some schedule a job is completed before or at its due date, then it is called *early* (or *on time*). The objective is to maximize the number of early jobs. For m identical machines, we prove an upper bound $1 - (1/2m)$ of competitive ratio and show that *ECT* (earliest completion time) algorithm is 1/2-competitive.

1. Introduction

In classical scheduling, it is generally assumed that the information for all jobs in an instance is known in advance. However, this assumption is not true in many situations. This promotes the emergence of online scheduling. There are three online models commonly considered [1]. The first model assumes that jobs arrive in a list without release dates. Any online algorithm has to schedule each job before seeing the next job in the list. The second model assumes that the running time of a job is unknown until the job is finished. The online algorithm only knows whether a job is still running at any time. The third model assumes that jobs arrive over time. At each time when a machine becomes idle, the algorithm schedules one of the available jobs, if any, on the machine.

In this paper, we consider the third model where jobs arrive over time. There are also three submodels for online scheduling where jobs arrive over time. The first one is *nonpreemptive model*, in which it is assumed that once a job is started on a machine, it must run to completion. The second one is *preemption-resume model*, in which it is assumed that a job on processing may be preempted at any time and be resumed from where it was preempted at a later moment. The third one is *preemption-restart model*. In this model, if a job is preempted during processing, it has to be restarted from the beginning for completion later on. Notice that, in the preemption-restart model, the finally completed jobs construct a nonpreemptive schedule. In this paper, we focus on the preemption-restart model in the online setting where jobs arrive over time.

We use the competitive analysis [2] to measure the performance of an online algorithm. For any job input sequence I, let $C_{\mathrm{ON}}(I)$ denote the number of early jobs in the schedule produced by an online algorithm $\mathcal{A}_{\mathrm{ON}}$, and let $C_{\mathrm{OPT}}(I)$ denote the number of early jobs in an optimal schedule. We say $\mathcal{A}_{\mathrm{ON}}$ is ρ-competitive where

$$\rho = \inf\left\{ \frac{C_{\mathrm{ON}}(I)}{C_{\mathrm{OPT}}(I)} \mid C_{\mathrm{OPT}}(I) > 0 \right\}. \tag{1.1}$$

ρ is also called the competitive ratio of $\mathcal{A}_{\mathrm{ON}}$. Clearly, $0 \leq \rho \leq 1$ holds. The closer the ratio ρ approaches 1, the better the performance of algorithm $\mathcal{A}_{\mathrm{ON}}$.

Sgall [3] gave a survey on online scheduling, including results on both nonpreemptive and preemption-resume models. Shmoys et al. [4] studied a preemption-restart model to minimize the makespan and presented several results for scheduling jobs on m parallel machines. Hoogeveen et al. [5] investigated the preemption-restart model in online single-machine scheduling to maximize the number of early jobs. They proved that the shortest remaining processing time (SRPT) rule yields an optimal online algorithm with competitive ratio 1/2. Note that SRPT rule implies earliest completion time (ECT) rule. In this paper, our main result is a kind of generalization of that in Hoogeveen et al. [5].

The rest of this paper is organized as follows. Section 2 introduces some definitions and notations. In Section 3, we show that $1 - (1/2m)$ is an upper bound of competitive ratio for all online algorithms. In Section 4, we present an online algorithm *ECT* based on the earliest completion time (ECT) rule and prove that the algorithm is 1/2-competitive.

2. Problem Definition and Notations

We are given m identical machines. Without loss of generality, we denote them by machine-1, ..., machine-m, respectively. Each machine processes at most one job at any time. A sequence of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ with due dates arrive over time where the value of n is unknown to online algorithms in advance. The information of each job is released on its release time to online algorithms. If a job is completed on or before its due date, we say it is an *early* job; otherwise it is a *tardy* job. The objective is to maximize the number of early jobs. We consider the preemption-restart model such that if a job on processing is preempted, then all the progress that has been made on the job so far is lost. A preempted job has to be restarted from the beginning to be completed later on.

Below we introduce some notations.

p_j: the processing time of job J_j.

r_j: the release time of job J_j.

d_j: the due date of job J_j.

Feasible schedule: a schedule consists of early jobs.

Current workload (of a machine): the total processing time of all jobs currently assigned to one machine at a time.

By the objective of maximizing the number of early jobs, we may schedule the processing of all tardy jobs after the last early job, and the processing of tardy jobs has no relation to the objective value. That is, it suffices to consider only feasible schedule and omit all the tardy jobs. In the remainder, when discussing a schedule produced by an online algorithm, we refer to a feasible schedule instead of the original schedule including preempted jobs.

At any time t, let $\overline{p_j}(t)$ denote the remaining processing time of job J_j with $r_j \le t$. From this definition, $\overline{p_j}(t) = p_j$ if J_j is neither on processing at time t nor completed by the time. On the other hand, if J_j was started at time s and is being processed throughout time interval $[s,t]$, then $\overline{p_j}(t) = p_j - (t-s)$. When the notation is completely understood, we sometimes use $\overline{p_j}$ instead of $\overline{p_j}(t)$ for notational convenience.

3. Upper Bound of Competitive Ratio

In order to show the upper bound of competitive ratio for all online algorithms, we use the following lemma.

Lemma 3.1. *In m identical machine scheduling problem, for any integer $k \ge 0$ and for all real numbers r and d with $r < d$, there exists an adversary strategy $\mathcal{S}_m = \mathcal{S}_m(k,r,d)$ with the following properties.*

(1) *$\mathcal{S}_m$ creates $(2k+1)m$ jobs. The earliest release time of these jobs is r, and the latest due date of these jobs is d.*

(2) *There exists a feasible schedule in which all the $(2k+1)m$ jobs are early. In such a schedule, the machines are continuously busy throughout interval $[r,d]$.*

(3) *If $u \in \{0,1,\ldots,m\}$ machines are unavailable throughout interval $[r,d)$, the adversary strategy $\mathcal{S}_m$ can prevent any online algorithm from scheduling more than $(2k+1)(m-u)$ jobs to be early.*

(4) *The adversary strategy $\mathcal{S}_m$ can prevent any online algorithm from scheduling more than $(2m-1)k+m$ jobs to be early. That is, any online algorithm has at least k tardy jobs.*

Proof. The proof is by induction on k. For $k = 0$, the adversary $\mathcal{S}_m$ releases m jobs with processing time $d-r$ at time r. For $k \ge 1$, the adversary $\mathcal{S}_m$ proceeds as follows. Let $L = (d-r)/8$, that is, $d = r+8L$. The adversary releases $2m$ jobs $J_1, J_2, \ldots, J_{2m}$ with processing times $p_i = 3L$ and $p_{m+i} = 4L$ for $i = 1,2,\ldots,m$ at time r. All these $2m$ jobs have due date d. Then the adversary waits until time $r+2L$.

Case 1. If at time $r+2L$ the online algorithm is processing at least one job J_i with $i \in \{1,2,\ldots,m\}$ and thus $p_i = 3L$, then $\mathcal{S}_m$ calls subadversary $\mathcal{S}_m(k-1, r+4L, r+5L)$.

Case 2. Otherwise, $\mathcal{S}_m$ calls subadversary $\mathcal{S}_m(k-1, r+3L, r+4L)$.

(1) The Proof of Property (1)

When $k = 0$, Property (1) holds. Consider the case $k \geq 1$. Assume that Property (1) follows for $k-1$. Since $\mathcal{S}_m$ creates $2m$ new jobs $J_1, J_2, \ldots, J_{2m}$ together with the $(2k-1)m$ jobs generated by the subadversary, Property (1) holds for k.

(2) The Proof of Property (2)

To prove Property (2), we consider the following schedules with all jobs early. In Case 1, machine-i $(1 \leq i \leq m)$ processes jobs J_i and J_{m+i} within intervals $[r+5L, d]$ and $[r, r+4L]$, respectively, then the m machines process all jobs of the subadversary by induction. In Case 2, machine-i $(1 \leq i \leq m)$ processes jobs J_i and J_{m+i} within intervals $[r, r+3L]$ and $[r+4L, d]$, respectively, then the m machines process all jobs of the subadversary by induction.

(3) The Proof of Property (3)

Given the assumption that $u \in \{0, 1, \ldots, m\}$ machines are unavailable throughout interval $[r, d]$, we prove Property (3) by induction on k. For $k = 0$, Property (3) holds, that is, $\mathcal{S}_m$ can prevent any online algorithm from scheduling more than $(m-u)$ jobs to be early with the assumption of u unavailable machines. Assume that for $k-1$ (in $k \geq 1$ case), Property (3) holds, that is, $\mathcal{S}_m(k-1, r, d)$ can prevent any online algorithm from scheduling more than $(2k-1)(m-u)$ jobs to be early with $m-u$ available machines. To prove that Property (3) holds for k, we observe that within interval $[r, d]$, $m-u$ available machines can schedule at most $2(m-u)$ jobs to be early among the $2m$ jobs that were released by $\mathcal{S}_m$. More precisely, the $m-u$ machines can process at most $m-u$ jobs during each of intervals $[r+3L, r+4L]$ and $[r+4L, r+5L]$. This observation is true for both Cases 1 and 2. Together with the $(2k-1)(m-u)$ jobs generated by the subadversary, there are at most $(2k+1)(m-u)$ early jobs, and thus Property (3) follows.

(4) The Proof of Property (4)

Now, we show that Property (4) holds by induction on k. First, it is a trivial case when $k = 0$. Assume that for $k-1$ (in $k \geq 1$ case), Property (4) holds, that is, $\mathcal{S}_m(k-1, r, d)$ can prevent any online algorithm from scheduling more than $(2m-1)(k-1)+m$ jobs to be early. The remainder is to prove that Property (4) holds for k. Consider the following two cases.

Case A. The online algorithm schedules all the $2m$ jobs $J_1, J_2, \ldots, J_{2m}$ to be early. If Case 1 happens, that is, at least one of the m jobs with length $3L$ is being processed at time $r+2L$. Notice that this job is to be completed no earlier than time $r+3L$. It implies that at least one of the m jobs, denoted by R, with length $4L$ must be started within interval $[r+3L, r+4L]$ to be completed on time due to its job length $4L$ and due date $d = r+8L$. The processing of R will then cover interval $[r+4L, r+5L]$ on one of the machines, that is, the machine for processing job R is unavailable to other jobs throughout the interval, then we call subadversary $\mathcal{S}_m = \mathcal{S}_m(k-1, r+4L, r+5L)$. By Property (3), throughout interval $[r+4L, r+5L]$, the online algorithm can schedule at most $(2k-1)(m-1)$ jobs to be early. Therefore, the total number of early jobs by the online algorithm is

$$2m + (2k-1)(m-1) = (2m-1)k + m + (1-k) \leq (2m-1)k + m. \tag{3.1}$$

Otherwise, if Case 2 happens, then all the m jobs, $J_{m+1}, \ldots, J_{2m}$, with length $4L$ must be completed by the m machines on or before time $d - 3L = r + 5L$. This implies that the m machines are unavailable for all jobs except jobs $J_k (m + 1 \leq k \leq 2m)$ within interval $[r + 3L, r + 4L]$. By calling subadversary $\mathcal{S}_m = \mathcal{S}_m(k - 1, r + 3L, r + 4L)$, the online algorithm schedules only $2m \leq (2m - 1)k + m$ jobs to be early. Hence, Property (4) holds in this case.

Case B. Among the $2m$ jobs $J_1, \ldots, J_{2m}$, the online algorithm schedules at most $2m - 1$ of them to be early. By induction, for $k - 1$, the online algorithm can schedule at most $(2m - 1)(k - 1) + m$ jobs to be early in both Case 1 and Case 2. Together with the other $\leq 2m - 1$ early jobs scheduled by the online algorithm for k, we have that the number of early jobs is at most

$$(2m - 1)(k - 1) + m + (2m - 1) = (2m - 1)k + m. \tag{3.2}$$

Property (4) follows. □

Remark 3.2. Lemma 2 in [5] is a special case of the above lemma with $m = 1$.

Theorem 3.3. *For the online scheduling problem on m identical machines to maximize the number of early jobs, any online algorithm $\mathcal{A}$ has competitive ratio $\rho_{\mathcal{A}} \leq 1 - (1/2m)$.*

Proof. Let C_{OPT} and C_{ON} denote the number of early jobs of an offline optimal algorithm (offline algorithm for short) and that of an online algorithm, respectively. By Lemma 3.1, we have $C_{\text{OPT}} = (2k + 1)m$ and $C_{\text{ON}} \leq (2m - 1)k + m$. Therefore,

$$\frac{C_{\text{ON}}}{C_{\text{OPT}}} \leq \frac{(2m - 1)k + m}{(2k + 1)m} \longrightarrow 1 - \frac{1}{2m}, \quad k \longrightarrow \infty. \tag{3.3}$$

The theorem follows. □

Remark 3.4. Theorem 3 in [5] is a special case of Theorem 3.3 with $m = 1$.

4. ECT Algorithm

In this section, we describe and analyze online algorithm *ECT*, which is based on the *shortest remaining processing time* (SRPT) or ECT rule. Algorithm *ECT* constructs a feasible schedule of early jobs only, since all tardy jobs can be appended to the end of this schedule in an arbitrary order.

Given a job instance $\mathcal{J}$, *algorithm ECT* runs as follows.

Step 1. Wait until a decision time point t, at which either a new job is released or at least one of the m machines, becomes idle.

Step 2. At time t, let J_i be the job such that $\overline{p_i} = \min\{\overline{p_k} \mid r_k \leq t, \ \overline{p_k} > 0 \text{ and } t + \overline{p_k} \leq d_k\}$, that is, J_i is with the shortest remaining processing time among all the uncompleted arrival jobs. If there is no idle machine at the time, schedule J_i on the machine with the least current workload and start the job immediately; otherwise, start to process J_i at the time on any idle machine.

Step 3. If at some decision point t, no more jobs are released and $t + \overline{p_k} > d_k$ holds for any arrival job J_k with $\overline{p_k} > 0$, stop; otherwise, go to Step 1.

Note that algorithm *ECT* applies the earliest completion time (ECT) policy to the case with m identical machines. By Step 2 of the algorithm, a job is preempted only if a newly released job is supposed to be completed earlier than the current one.

Without loss of generality, we assume that the m machines are reindexed in nondecreasing order of job length considering the m jobs which are scheduled at the first position on the m machines. Let $N(S)$ be the number of early jobs in a schedule S. Let $\vec{S}$ be the feasible schedule produced by *ECT*, and let S^* be an optimal schedule, respectively. Let $J_k(S)$ be the job with the kth smallest completion time in schedule S. If two jobs are completed at the same time, the job completed on the machine with smaller index is regarded to be completed earlier.

Note that $\vec{S}$ only contains early jobs and algorithm *ECT* terminates at some time t, at which none of the rest uncompleted jobs in $\mathcal{J}$ can meet its due date even if started at once.

Theorem 4.1. *Algorithm ECT is* 1/2*-competitive.*

Proof. If $S^* = \vec{S}$, the theorem follows. In the following proof, we assume that $S^* \neq \vec{S}$. To prove the theorem, it is sufficient to prove that $N(\vec{S}) \geq N(S^*)/2$. We construct a series of feasible schedules $S^0, S^1, \ldots, S^h$ such that $S^0 = S^*$, $S^h = \vec{S}$, and $S^q (q = 1, \ldots, h-1)$ is different from $\vec{S}$. S^q is obtained from S^{q-1} for $q = 1, \ldots, h$ in the following way.

Let $k'(q)$ be the smallest k such that $J_k(S^{q-1}) \neq J_k(\vec{S})$ for $1 \leq k \leq N(\vec{S})$. Such a k does exist since S^{q-1} is different from $\vec{S}$. From the notation, we have either $k'(q) = 1$ or

$$J_k\left(S^{q-1}\right) = J_k\left(\vec{S}\right), \quad k \in \{1, \ldots, k'(q) - 1\}. \tag{4.1}$$

S^q is obtained by

(i) deleting job $J_{k'(q)}(S^{q-1})$ from S^{q-1};

(ii) either moving or adding job $J_{k'(q)}(\vec{S})$ to the place of $J_{k'(q)}(S^{q-1})$, depending on whether the job was already in S^{q-1}. □

By construction, S^q is necessarily feasible since $J_{k'(q)}(\vec{S})$ is the job to be completed the earliest among all uncompleted jobs at that time (ECT policy). Furthermore, for $q = 1, \ldots, h$, we have either $S^q = \vec{S}$ or

$$k'(q) > k'(q-1), \tag{4.2}$$

$$N(S^q) \geq N\left(S^{q-1}\right) - 1. \tag{4.3}$$

From inequality (4.2), we obtain

$$h \leq N\left(\vec{S}\right). \tag{4.4}$$

From inequalities (4.3) and (4.4), we obtain

$$N\left(\vec{S}\right)=N\left(S^{h}\right)\geq N\left(S^{h-1}\right)-1\geq N\left(S^{h-2}\right)-2\geq\cdots\geq N\left(S^{0}\right)-h=N(S^{*})-h\geq N(S^{*})-N\left(\vec{S}\right). \tag{4.5}$$

As a consequence, $N(\vec{S})\geq N(S^{*})/2$, and the theorem follows.

Acknowledgment

This work is partially supported by NSF of China under Grants nos. 71172189, 71101106, 70832005, 71071123, and 71090404/71090400.

References

[1] K. Pruhs, J. Sgall, and E. Torng, "Online scheduling," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J. Y.-T. Leung, Ed., 2004.

[2] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[3] J. Sgall, "On-line scheduling," in *On-line Algorithms: The State of the Art*, A. Fiat and G. J. Woeginger, Eds., vol. 1442 of *Lecture Notes in Computer Science*, pp. 196–231, Springer, Berlin, Germany, 1998.

[4] D. B. Shmoys, J. Wein, and D. P. Williamson, "Scheduling parallel machines on-line," *SIAM Journal on Computing*, vol. 24, pp. 1313–1331, 1995.

[5] H. Hoogeveen, C. N. Potts, and G. J. Woeginger, "On-line scheduling on a single machine: maximizing the number of early jobs," *Operations Research Letters*, vol. 27, no. 5, pp. 193–197, 2000.