Scheduling Jobs Online on Linear Hierarchical Parallel Machines

Xiao XIN*,1, Min MOU1, Guohua MU1

College of Foreign Studies, Shandong Institute of Business and Technology, Yantai, 264005, China

***Abstract* — A parallel machine scheduling problem is considered in the presence of machine eligibility restrictions. The machines run at the same constant speed but they form a linear hierarchy of capability, i.e., leftward machines are more capable than rightward ones. Jobs arrive online over time at their release times. The processing time of a job will be known immediately upon its arrival. Each job specifies the least capable machine that can process it, and this job can be assigned to any of the machines to the left of this machine (including this machine itself). The jobs must be processed non-preemptively. The objective is to minimize the maximum flow time, where the flow time of a job is defined to be the difference between its release time and completion time. A constant competitive ratio algorithm for the problem is presented. Previously, no such result was known even in the offline setting.**

Keywords - online scheduling; parallel machines; machine eligibility constraints; linear hierarchy; maximum flow time; constant competitive ratio

I. INTRODUCTION

The scheduling of a set of jobs on parallel machines is a well-studied problem in combinatorial optimization, with a number of increasingly important applications in practice [1, 2, 3]. The overwhelming majority of the results in the literature assume that each machine is capable of processing all the jobs. This fails, however, to capture many real-world situations where a job can only be processed on a subset of the parallel machines. For example, a modern machine with new technologies may process some jobs which cannot be processed on the machines with older technologies [4]. Another example exists in the service industry where various customers are entitled to many different grades of service levels [5].

Research on the scheduling problem with machine eligibility restrictions focuses on developing heuristics for various objectives due to the complexity of this problem. Centeno and Armacost [6] presented an algorithm for the problem of minimizing maximum lateness with release times and machine eligibility restrictions for the special case where due dates are equal to release times plus a constant. Later, they [4] studied the online scheduling problem of minimizing makespan with release times and machine eligibility restrictions, and demonstrated that when the machine eligibility sets are not nested, the longest processing time rule performs better than the least flexible job rule in the presence or absence of release time stipulations. C.H. Lin and C.J. Liao [7] considered a two level scheduling problem which is a special case of the problem with machine eligibility restrictions. Both machines and jobs can be classified into high and low levels, and each job can only be processed by the machine whose level is no more than that of the job. This problem is already NP-hard [5] and C.H. Lin and C.J. Liao [7] presented an optimal algorithm which is essentially based on enumeration and thus has exponential time complexity. R. Gokhale and M. Mathirajan [8] studied the problem of minimizing total weighted flow time in the presence of release times, sequence-dependent setup times, and machine eligibility restrictions. They developed a mathematical model for this problem and presented five heuristic algorithms.

For the load balancing problem with machine eligibility restrictions, Azar et al. [9] proved that the competitive ratio of any online algorithm is $\Omega(\sqrt{m})$, where m denotes the number of machines. An algorithm with a matching upper bound was derived by Azar et al. [10]. Since arbitrary assignment restriction makes the problem significantly harder, it is natural to consider some special cases where the assignment restriction is allowed in a more controllable manner. In particular, Bar-Noy et al. [11] initiated the study of the following hierarchical model. The machines form a hierarchy of capability; each job requests a specific machine meeting its needs, but the system is free to assign it either to that machine or to any other machine higher in the hierarchy. They presented constant competitive ratio algorithms for the linear hierarchy. They also presented constant competitive ratio algorithms for the linear hierarchy where only permanent jobs are allowed, and showed an $\Omega(\sqrt{m})$ lower bound when temporary jobs are allowed.

To the best of our knowledge, the problem of minimizing maximum flow time with machine eligibility restrictions has not been studied to date, even for the offline setting. The flow time of a job is defined to be the difference between its release time and completion time. The minimization of maximum flow time guarantees that each job has a small delay. For the problem of minimizing maximum flow time without machine eligibility, the first known result is due to Bender et al. [12]. They presented a $(3-2/m)$ -competitive algorithm for the online non-preemptive problem with identical parallel machines. If preemption is allowed, C. Ambühl and M. Mastrolilli [13] derived an optimal online algorithm with competitive ratio $2-1/m$. Recently, N. Bansal and B. Cloostermans [14] obtained the first constant competitive ratio algorithm for the problem of minimizing maximum flow time on related machines.

Motivated by [11] and [14], we consider the problem of online scheduling linear hierarchical parallel machines to minimize maximum flow time, and present a constant competitive ratio algorithm for it by modifying the algorithm presented in [14].

The remainder of this paper is organized as follows. In Section 2, we formally define the problem considered, and review terminology needed. In Section 3, we describe the algorithm. Section 4 analyses the algorithm's competitive ratio. We conclude this paper in Section 5.

II. PRELIMINARIES

In the *hierarchical machines* problem the machines form a hierarchy of capability; a job which can be processed on a given machine can also be processed on any machine higher in the hierarchy. The *linear hierarchy* means that the machines are numbered 1 through m, ordered along a straight line running from left to right, with machine 1 leftmost and machine m rightmost. Leftward machines are more capable than rightward ones. We say, as in [11], that the machines $1,2,\ldots,s$ are *to the left* of s, and the machines $s+1,\ldots,m$ are *to the right* of s.

Each job j ($j=1,2,\ldots,n$) is released at time $r_j \geq 0$ and cannot start processing before that time. Job j also has a *processing time* p_j and requests one of the machines. A job requesting machine s can be assigned to any of the machines to the left of s which are the job's *eligible machines*. Each machine can process at most one job at a time, and each job must be processed in a non-preemptive fashion on one of its eligible machines. Let C_j be the *completion time* of job j in a given schedule. Then the *flow time* of job j is defined as $F_j = C_j - r_j$, and the *maximum flow time* of the schedule is $F_{\max} = \max_{j=1,2,\ldots,n} F_j$. The objective is to find a schedule that minimizes the maximum flow time.

In the *offline* version of the problem, it is assumed that a complete specification of the problem instance is available before the algorithm begins to construct a schedule. By contrast, in the *online* version of the problem, jobs arrive over time, all job characteristics become known at their release times, and the algorithm must decide only based on already arrived jobs [2].

Online algorithms can be evaluated in terms of their *competitive ratio* [15], which is defined as the maximum ratio, taken over all possible problem instances, of the objective value achieved by an online algorithm to the objective value achieved by an optimal offline algorithm.

III. THE ALGORITHM

In this section we will present a constant competitive ratio algorithm EpochSchedule for the problem of online scheduling linear hierarchical parallel machines to minimize maximum flow time. This also gives the first constant approximation for the offline problem.

Let *Opt* be an optimal offline schedule. We also use *Opt* to denote the objective value of this schedule. Let F_{opt} be an estimate of *Opt*. In the end of next section we will show how to ensure $F_{opt} \in [Opt,\ 1.5Opt]$.

We partition the time interval $[0,\ \infty)$ into disjoint intervals I_k of size $3F_{opt}$ as $I_k = [3(k-1)F_{opt},\ 3kF_{opt})$. The time $3kF_{opt}$ is referred to as the k-th *epoch*. If at time t the total remaining processing time of jobs on machine i is $l_i(t)$, we say that machine i has load $l_i(t)$ at time t. Denote by $[1:i]$ the machines $1,2,\ldots,i$. For each $k=1,2,\ldots$, EpochSchedule schedules the jobs arriving during I_k at epoch k using the following algorithm.

Algorithm EpochSchedule for the epoch *k*:

Step 1. Partition the jobs arriving during I_k into classes $J_1,\ldots,J_m$, where each job j is in class J_i if and only if machine i is the rightmost eligible machine for j.

Step 2. For $i=1,2,\ldots,m$, assign the jobs j in J_i in arbitrary order as follows:

(**Saturation phase**:) **If** some machine in $[1:i]$ is loaded below $3F_{opt}$ schedule j on the rightmost such machine.

(**Rightmost-fit phase**:) **Else** schedule j on the rightmost machine in $[1:i]$ such that its load stays below $6F_{opt}$.

Step 3. If no such machine exists return FAIL.

In the next section, we will prove the next theorem.

Theorem 1. *If* $F_{opt} \geq Opt$, *then EpochSchedule never loads machine 1 above* $5F_{opt}$ *and thus never fails.*

Based on this theorem, we can get our main result.

Theorem 2. *EpochSchedule is a 13.5-competitive algorithm for the problem of online scheduling linear hierarchical parallel machines to minimize maximum flow time.*

Proof. The flow time of any job is at most $3F_{opt} + 6F_{opt} = 9F_{opt}$, since a job has to wait at most

$3F_{opt}$ to be assigned, and spends at most $6F_{opt}$ on its designated machine. Recall that $F_{opt} \in [Opt,\ 1.5Opt]$, hence we get the desired competitive ratio.

IV. ANALYSIS

In this section we will prove Theorem 1, and show how to ensure $F_{opt} \in [Opt,\ 1.5Opt]$.

Bender et al. [12] showed that the Fist-In-First-Out (FIFO) strategy (that is, scheduling jobs in the order they arrive) is optimal for minimizing maximum flow time on a single machine. Consider an optimal schedule in which FIFO strategy is applied on each machine. By delaying every job by at most $3F_{opt}$, we can transform this schedule into a *restricted* one in which the jobs arriving during I_k are scheduled at epoch k , just as EpochSchedule. This restricted schedule has objective value at most $3F_{opt} + Opt \le 4F_{opt}$, if $F_{opt} \ge Opt$. We call this schedule *ropt schedule*.

Fix an epoch k . Let $A_i(k)$ be the total load on machines $[1:i]$ at time $3kF_{opt}$ in EpochSchedule just before the jobs arriving during I_k are scheduled. Let $B_i(k)$ be the total load on machines $[1:i]$ at time $3kF_{opt}$ in EpochSchedule just after all the jobs arriving during I_k are scheduled. Let $A_i^{ropt}(k)$ be the total load on machines $[1:i]$ at time $3kF_{opt}$ in ropt schedule just before the jobs arriving during I_k are scheduled. Let L_i^{ropt} be the load on machine i in ropt schedule just after all the jobs arriving during I_k are scheduled. Let $B_i^{ropt}(k) = \sum_{i'=1}^{i} \max\{L_{i'}^{ropt}, 3F_{opt}\}$. We will prove that the following two invariants hold at every epoch k for all $1 \le i \le m$.

$$A_i(k) \le A_i^{ropt}(k) + i \cdot F_{opt} \qquad (1)$$

$$B_i(k) \le B_i^{ropt}(k) + i \cdot F_{opt} \qquad (2)$$

In order to prove (1) and (2), we need the following two lemmas.

Lemma 1. *If at epoch k , (1) holds for $1 \le i \le m$, then (2) holds for $1 \le i \le m$.*

Lemma 2. *If at epoch k , (2) holds for $1 \le i \le m$, then (1) holds for $1 \le i \le m$ at epoch $k+1$.*

Since $A_i(0) = A_i^{ropt}(0)$ for all $1 \le i \le m$, the invariant (1) holds trivially for $k = 0$. Hence we can apply Lemmas 1 and 2 alternately to prove (1) and (2) for all k .

A. Proof of Lemma 1

Lemma 3. *Let $i_1 > i_2$. If a job j in J_{i_1} is scheduled by EpochSchedule onto machine i_2 during the saturation phase, then all jobs in J_i for $i_2 \le i < i_1$ are also scheduled by EpochSchedule during the saturation phase.*

Proof. Since j is scheduled on i_2 during the saturation phase, the load on i_2 is below $3F_{opt}$ before j is scheduled. Jobs in J_i for $i_2 \le i < i_1$ are considered before J_{i_1} , therefore the load on i_2 is below $3F_{opt}$ after J_i is considered. It follows that all jobs in J_i for $i_2 \le i < i_1$ are scheduled by EpochSchedule during the saturation phase. □

Definition 1. Let $i_1 > i_2$. Machines i_1 and i_2 are *separated* at epoch k if no jobs from classes $[i_1 : m]$ are scheduled by EpochSchedule onto machines $[1 : i_2]$ at epoch k .

Lemma 4. *If machines $i+1$ and i are separated at epoch k , then (1) implies (2) fior machine i .*

Proof. Since machines $i+1$ and i are separated, no jobs from classes $[i+1:m]$ are scheduled onto machines $[1:i]$ at epoch k . Let $|J_{i'}|$ denote the total processing time of jobs in $J_{i'}$. It follows that $B_i(k) = A_i(k) + \sum_{i'=1}^{i} |J_{i'}|$.

On the other hand, in any feasible schedule, jobs in $J_{i'}$ *for* $1 \le i' \le i$ cannot be assigned onto machines $[i+1:m]$. It follows that $B_i^{ropt}(k) \ge A_i^{ropt}(k) + \sum_{i'=1}^{i} |J_{i'}|$.

Since $A_i(k) \le A_i^{ropt}(k) + i \cdot F_{opt}$, we get $B_i(k) \le A_i(k) + B_i^{ropt}(k) - A_i^{ropt}(k) \le B_i^{ropt}(k) + i \cdot F_{opt}$. We use induction over i to prove Lemma 1. Since all relevant quantities are zero for $i = 0$, the base case is trivially true.

There are three different cases depending on how jobs from classes $[i+1:m]$ are scheduled onto machines $[1:i]$ at epoch k by EpochSchedule.

Case 1. No jobs at all from classes $[i+1:m]$ are scheduled onto machines $[1:i]$ at epoch k .

In this case, machines $i+1$ and i are separated. By Lemma 4, the invariant (2) holds.

Case 2. Jobs from classes $[i+1:m]$ are scheduled onto machines $[1:i]$ at epoch k only during the saturation phase.

Let $i_{\max} \le i$ be the largest index such that machine s $i+1$ and $i_{\max} - 1$ are separated (if $i_{\max}$ does not exist, then set $i_{\max} = 1$). By the inductive hypothesis, (2) holds for $i_{\max} - 1$ and hence we get $B_{i_{\max}-1}(k) \le B^{ropt}_{i_{\max}-1}(k) + (i_{\max} - 1) \cdot F_{opt}$. Since jobs from classes $[i+1:m]$ are scheduled onto machines $[i_{\max} : i]$ at epoch k only during the saturation phase, Lemma 3 ensures that jobs from classes $[i_{\max} : i]$ are also scheduled during the saturation phase. It follows that the load of each machine in $[i_{\max} : i]$ stays below $4F_{opt}$. Hence we get

$$\begin{aligned} B_i(k) &\le 4F_{opt}(i - i_{\max} + 1) + B_{i_{\max}-1}(k) \\ &\le 4F_{opt}(i - i_{\max} + 1) + B^{ropt}_{i_{\max-1}}(k) + (i_{\max} - 1) \cdot F_{opt} \\ &\le 3F_{opt}(i - i_{\max} + 1) + B^{ropt}_{i_{\max-1}}(k) + i \cdot F_{opt} \\ &\le B^{ropt}_i(k) + i \cdot F_{opt}. \end{aligned}$$

Case 3. Some job from classes $[i+1:m]$ is scheduled onto machines $[1:i]$ at epoch k during the rightmost-fit phase.

In this case, it must be true that machine $i+1$ has load more than $5F_{opt}$. Let $i_{\min} \ge i+1$ be the index such that machines $[i+1:i_{\min}]$ have load more than $5F_{opt}$ and machine $i_{\min}+1$ has load at most $5F_{opt}$. If $i_{\min}$ does not exist, then set $i_{\min} = m$. Suppose that some job j from classes $[i_{\min}+1:m]$ is scheduled onto machines $[1:i_{\min}]$. Since machine $i_{\min}+1$ has load at most $5F_{opt}$, job j cannot be assigned during the rightmost-fit phase. Since machine $i_{\min}$ has load more than $5F_{opt}$, job j cannot be assigned during the saturation phase. This contradiction shows that machines $i_{\min}+1$ and $i_{\min}$ are separated.

Applying Lemma 4 to $i_{\min}$, we get

$$B_{i_{\min}}(k) \le B^{ropt}_{i_{\min}}(k) + i_{\min} \cdot F_{opt} \tag{3}$$

Since all the machines $[i+1:i_{\min}]$ have load more than $5F_{opt}$, we get

$$B_i(k) \le B_{i_{\min}}(k) - 5F_{opt} \cdot (i_{\min} - i) \tag{4}$$

Since in the ropt schedule every machine has load at most $4F_{opt}$, we get

$$B^{ropt}_{i_{\min}}(k) \le B^{ropt}_i(k) + 4F_{opt} \cdot (i_{\min} - i) \tag{5}$$

Combining the inequalities (3), (4) and (5), we get:

$$\begin{aligned} B_i(k) &\le B^{ropt}_{i_{\min}}(k) + i_{\min} \cdot F_{opt} - 5F_{opt} \cdot (i_{\min} - i) \\ &\le B^{ropt}_i(k) + i_{\min} \cdot F_{opt} - F_{opt} \cdot (i_{\min} - i) \\ &= B^{ropt}_i(k) + i \cdot F_{opt}, \end{aligned}$$

which completes the proof of Lemma 1. □

B. *Proof of Lemma 2*

We use induction over i to prove Lemma 2. Since all relevant quantities are zero for $i = 0$, the base case is trivially true. There are two different cases depending on the value of $B_i(k) - B_{i-1}(k)$.

Case 1. $B_i(k) - B_{i-1}(k) \le 4F_{opt}$.

In this case, we have $A_i(k+1) - A_{i-1}(k+1) \le F_{opt}$. By the inductive hypothesis, we have $A_{i-1}(k+1) \le A^{ropt}_{i-1}(k+1) + (i-1) \cdot F_{opt}$. Hence we get

$$\begin{aligned} A_i(k+1) &\le A_{i-1}(k+1) + F_{opt} \\ &\le A^{ropt}_{i-1}(k+1) + (i-1) \cdot F_{opt} + F_{opt} \\ &\le A^{ropt}_{i-1}(k+1) + i \cdot F_{opt} \\ &\le A^{ropt}_i(k+1) + i \cdot F_{opt}, \end{aligned}$$

where the last inequality follows as $A^{ropt}_i(k+1)$ is non-decreasing as i increases.

Case 2. $B_i(k) - B_{i-1}(k) > 4F_{opt}$.

In this case, EpochSchedule must have assigned a job onto machine i during the rightmost-fit phase. It follows that all the machine $[1:i-1]$ have load more than $3F_{opt}$. Thus we get $A_i(k+1) = B_i(k) - i \cdot 3F_{opt}$.

On the other hand, recall that $B^{ropt}_i(k)$ is defined as $\sum_{i'=1}^{i} \max\{L^{ropt}_{i'}, 3F_{opt}\}$. It follows that $A^{ropt}_i(k+1) = B^{ropt}_i(k) - i \cdot 3F_{opt}$. We get

$$\begin{aligned} A_i(k+1) &= B_i(k) - i \cdot 3F_{opt} \\ &\le B^{ropt}_i(k) + i \cdot F_{opt} - i \cdot 3F_{opt} \\ &= A^{ropt}_i(k+1) + i \cdot F_{opt}, \end{aligned}$$

and hence (1) holds for i at epoch $k+1$, which completes the proof of Lemma 2. □

C. *Proof of Theorem 1*

Recall that in the ropt schedule the jobs arriving during I_k are scheduled at epoch k , just as EpochSchedule. This restricted schedule has objective value at most $4F_{opt}$, if $F_{opt} \ge Opt$. It follows that $B^{ropt}_i(k) \le i \cdot 4F_{opt}$.

Therefore we get $B_i(k) \le i \cdot 5F_{opt}$ by the invariant (2). Choosing $i = 1$, we conclude that if $F_{opt} \ge Opt$, then EpochSchedule never loads machine 1 above $5F_{opt}$ and thus machine 1 can always accommodate an additional job.

D. *Estimating the optimal value*

During the run of EpochSchedule, if it fails at epoch k , we know that $F_{opt} < Opt$ by Theorem 1. We then cancel the assignment of the jobs arriving during I_k . Set $F'_{opt} = 1.5F_{opt}$ and let the k -th epoch to be the time $(k-1)F_{opt} + 3F'_{opt}$. We reschedule the jobs arriving during I_k at the new epoch k . The load of any machine at the next epoch will be at most $6F_{opt} - 3F'_{opt} = F'_{opt}$. This ensures that (1) holds for all i , i.e., $A_i(k) \le i \cdot F'_{opt} \le A_i^{ropt}(k) + i \cdot F'_{opt}$. If $F'_{opt} \ge Opt$, then (2) holds for all i and EpochSchedule proceeds as normal.

V. CONCLUSION

In this paper we investigated the problem of online scheduling linear hierarchical parallel machines to minimize maximum flow time, and presented a constant competitive ratio algorithm for it. This also gives the first constant approximation for the offline problem. It would be interesting to consider other hierarchical machine topologies, such as trees, and try to get constant competitive ratio algorithms.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Nos. 61272430 and 61472227).

REFERENCES

[1] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," Annals of discrete mathematics, vol. 5, pp. 287-326, 1979.

[2] J. Y. Leung, Handbook of scheduling: algorithms, models, and performance analysis: CRC Press, 2004.

[3] Y. Robert and F. Vivien, Introduction to scheduling: CRC Press, 2009.

[4] G. Centeno and R. L. Armacost, "Minimizing makespan on parallel machines with release time and machine eligibility restrictions," International Journal of Production Research, vol. 42, pp. 1243-1256, 2004.

[5] H.-C. Hwang, S. Y. Chang, and K. Lee, "Parallel machine scheduling under a grade of service provision," Computers & Operations Research, vol. 31, pp. 2055-2061, 2004.

[6] G. Centeno and R. L. Armacost, "Parallel machine scheduling with release time and machine eligibility restrictions," Computers & industrial engineering, vol. 33, pp. 273-276, 1997.

[7] C.-H. Lin and C.-J. Liao, "Minimizing makespan on parallel machines with machine eligibility restrictions," Open Operational Research Journal, vol. 2, pp. 18-24, 2008.

[8] R. Gokhale and M. Mathirajan, "Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing," The International Journal of Advanced Manufacturing Technology, vol. 60, pp. 1099-1110, 2012.

[9] Y. Azar, A. Z. Broder, and A. R. Karlin, "On-line load balancing," Theoretical Computer Science, vol. 130, pp. 73-84, 1994.

[10] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. R. Pruhs, and O. Waarts, "On-line load balancing of temporary tasks," Journal of Algorithms, vol. 22, pp. 93-110, 1997.

[11] A. Bar-Noy, A. Freund, and J. Naor, "On-line load balancing in a hierarchical server topology," SIAM Journal on Computing, vol. 31, pp. 527-549, 2001.

[12] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and Stretch Metrics for Scheduling Continuous Job Streams," in SODA, 1998, pp. 270-279.

[13] C. Ambühl and M. Mastrolilli, "On-line scheduling to minimize max flow time: an optimal preemptive algorithm," Operations Research Letters, vol. 33, pp. 597-602, 2005.

[14] N. Bansal and B. Cloostermans, "Minimizing maximum flow-time on related machines," in LIPIcs-Leibniz International Proceedings in Informatics, 2015.

[15] Sleator, Daniel D., and Robert E. Tarjan. "Amortized efficiency of list update and paging rules," Communications of the ACM, vol. 28(2), pp. 202-208, 1985.