Journal of Scheduling (2018) 21:647–654
https://doi.org/10.1007/s10951-018-0556-2

Online scheduling of moldable parallel tasks

Deshi Ye[1] · Danny Z. Chen[2] · Guochuan Zhang[1]

Published online: 2 February 2018

Abstract
In this paper, we study an online scheduling problem with moldable parallel tasks on m processors. Each moldable task can be processed simultaneously on any number of processors of a parallel computer, and the processing time of a moldable task depends on the number of processors allotted to it. Tasks arrive one by one. Upon arrival of each task, the scheduler has to determine both the number of processors and the starting time for the task. Moreover, these decisions cannot be changed in the future. The objective is to attain a schedule such that the longest completion time over all tasks, i.e., the makespan, is minimized. First, we provide a general framework to show that any ρ-bounded algorithm for scheduling of rigid parallel tasks (the number of processors for a task is fixed a prior) can be extended to yield an algorithm for scheduling of moldable tasks with a competitive ratio of 4ρ if the ratio ρ is known beforehand. As a consequence, we achieve the first constant competitive ratio, 26.65, for the moldable parallel tasks scheduling problem. Next, we provide an improved algorithm with a competitive ratio of at most 16.74.

Keywords Online scheduling · Moldable tasks · Multi-core scheduling · Competitive analysis

1 Introduction

Virtualization technology is enriching the research on parallel task scheduling. For example, Kalé (2002) studied virtualization benefits for Charm++ and AMPI systems. In virtualization, a parallel task can be processed on a smaller number of processors than requested, though the processing time of the task may increase due to the influence of communications, synchronization, and other overheads. A task is *moldable* if the degree of parallelism (the number of processors used to process it) can be chosen by a scheduler when the task arrives (but the number of processors used for this task, once decided, cannot be changed during the whole processing). A task is *rigid* if the number of processors for processing it is fixed a priori. A task is *malleable* if the number of processors may change during its processing by preemption of tasks or simply by data redistributions. Note that some literature also names moldable tasks as *malleable* tasks (Ludwig and Tiwari 1994; Mounié et al. 2007).

Dutot et al. (2004) pointed out that most parallel applications are moldable. Cirne and Berman (2001) provided a model to generate moldable tasks from rigid workloads. In some real applications, tasks may arrive online, such as parallel short sequence mapping in DNA sequencing analysis (Saule et al. 2010). The concept of moldable tasks offers a powerful way to model large-scale parallel applications. But, it also adds an additional dimension of consideration to scheduling, i.e., the scheduler has to decide how many processors are used for each task.

In this paper, we study online scheduling of moldable tasks on a parallel computer system. We are given a set of n independent moldable jobs and m identical processors. An instance of the problem consists of a set of n jobs, $J = \{J_1, J_2, \ldots, J_n\}$, and a processing time function $p_{j,s}$ that represents the processing time of job J_j when executed on s processors. The scheduler shall assign a subset of S_j processors for each job J_j and find a starting time t_j such that job J_j starts its execution simultaneously at time t_j on all these processors. In this work, we consider the online model, in which jobs are released one by one, and the scheduler shall

✉ Deshi Ye
yedeshi@zju.edu.cn

Danny Z. Chen
dchen@nd.edu

Guochuan Zhang
zgc@zju.edu.cn

[1] College of Computer Science, Zhejiang University, Hangzhou 310027, China

[2] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA

make the decision for each job without being aware of any future jobs, and this decision cannot be changed later. The goal of the scheduler is to minimize the *makespan*, defined as the longest completion time over all the jobs, i.e., to find a feasible schedule minimizing $C_{\max} = \max_{j=1,\ldots,n}\{t_j + p_{j,|S_j|}\}$. Henceforth, the terms of processor and machine, and the terms of task and job, are interchangeable, respectively.

We use competitive analysis (Borodin and El-Yaniv 1998) to measure the performance of an online algorithm. For any input instance I and an online algorithm A, we denote by $OPT(I)$ and $A(I)$, respectively, the makespan of an optimal offline algorithm and the makespan of the algorithm A to schedule the instance I. Algorithm A is said to be ρ-competitive if $A(I) \le \rho \cdot OPT(I)$ for any instance I. The *competitive ratio* of algorithm A is defined as

$$R_A = \sup_I \{A(I)/OPT(I)\}.$$

The reader might wonder whether a simple greedy algorithm is good enough to solve this online moldable task scheduling problem: Upon the arrival of each job J_j, simply use all m processors to process it, and let J_j start the execution once the execution of J_{j-1} is finished. However, this greedy algorithm can suffer an arbitrarily large competitive ratio. The following is a simple example to illustrate this. There are m jobs, all of which have processing time 1 on 1 processor and $1-\epsilon$ on m processors, where ϵ is an arbitrarily small positive constant. It is easy to see that the makespan of this greedy algorithm is $m(1-\epsilon)$, while the optimal makespan is 1.

1.1 Related work

Offline moldable task scheduling has been studied extensively in the past decades; the reader is referred to the excellent survey (Dutot et al. 2004). Moldable tasks are called *monotone* if the processing time of any task is non-increasing in the number of machines and the work (the number of machines times the processing time) is non-decreasing in the number of machines. Belkhale and Banerjee (1990) presented a $2/(1+1/m)$-approximation algorithm under the monotonic assumption, where m is the number of total processors.

A basic approach for moldable jobs scheduling consists of two steps. The first step determines the number of machines for each job and the second step solves the resulting rigid task scheduling problem. In this approach, Turek et al. (1992) showed a 2-approximation algorithm without the monotonic assumption; the running time of their algorithm is $O(mnL(m,n))$, where m is the number of machines, n is the number of jobs, and $L(m,n)$ is the running time of an algorithm for the rigid jobs scheduling problem. Later, Ludwig and Tiwari (1994) presented an algorithm with the same approximation ratio of 2 without the monotonic assumption, but with an improved running time $O(mn + L(m,n))$.

Another approach for moldable jobs scheduling is to use dual approximation. This technique first selects an objective value and then decides the most efficient number of machines for each job such that all the jobs can be completed within this selected value. Under the *monotonic* assumption, Mounié et al. (1999) gave a $(\sqrt{3}+\varepsilon)$-approximation algorithm. Later, they improved the ratio to $(3/2+\varepsilon)$ (Mounié et al. 2007). Recently, Jansen (2012) also gave an algorithm with an approximation ratio of $3/2+\varepsilon$, but without the monotonic assumption.

Some special cases yield better results. When the number m of machines is a constant, Jansen and Porkolab (2002) proposed a linear time PTAS. When the number of machines is polynomially bounded by the number of jobs, Jansen and Thöle (2010) gave a PTAS. For the case of identical moldable jobs, Decker et al. (2006) presented a 5/4-approximation algorithm.

Note that the results mentioned above are not for *online* moldable task scheduling. To the best of our knowledge, no previous work is known for the general online moldable task scheduling. However, some special cases have been investigated. Rapine et al. (1998) considered scheduling online preemptive moldable jobs with the monotonic assumption, and the competitive ratios derived are not constant. In their model, the number of machines allocated to a job can be changed anytime during the execution, and job preemption is allowed. Dutton and Mao (2007) dealt with a model in which the processing time of a job J_j is $t_j = p_j/k_j + (k_j - 1)\cdot c$, where c is a constant, and job J_j has a processing requirement p_j and is assigned to k_j machines. They gave competitive ratios of earliest completion time (ECT) algorithms for $m \le 4$ machines and presented a general lower bound of 2.3 for a large number of machines. For a larger number of machines, an upper bound of around 4 of shortest execution time (SET) algorithm was presented in Havill and Mao (2008). Recently, Kell and Havill (2015) presented improved results for two and three machines. Guo and Kang (2010) extended this model for two machines, in which a job-dependent overhead term c_j replaces a uniform constant c.

The strip packing problem is to minimize the height required for placing a set of rectangular items into a strip with fixed width and unbounded height. Rigid tasks scheduling is closely related to the classic strip packing problem, in which it is in addition required to allocate consecutive processors to a task. Regarding online strip packing problems, Baker and Schwartz (1983) developed the first fit shelf algorithm with a competitive ratios of 6.99. Later, improved algorithms were proposed with a competitive ratio of 6.66 in two independent results (Hurink and Paulus 2007; Ye et al. 2009). Yu et al. (2016) provided a lower bound of 2.618.

1.2 Our results

In this paper, we study the online moldable tasks scheduling problem. The processing time of a moldable task is a function of the number of processors allocated to it. While previous studies focused on some restricted functions of processing time, we consider the general function without any assumption. Our first result is to provide a general framework, in which we present a black-box reduction such that any online moldable task scheduling algorithm is reduced to a rigid parallel task scheduling algorithm. Our work is motivated by the doubling technique (e.g., Aspnes et al. 1997) for online algorithms. In particular, we define a ρ-bounded algorithm (the detailed definition is given in Sect. 2) and then show that a 4ρ-competitive algorithm for the moldable task scheduling problem can be obtained from any ρ-bounded algorithm for the rigid parallel task scheduling problem. Since the algorithms in Baker and Schwartz (1983), Hurink and Paulus (2007), Ye et al. (2009) for rigid parallel task scheduling are ρ-bounded algorithms, we obtain a 26.65-competitive algorithm for the online moldable tasks scheduling problem. Note that the knowledge of ρ is required. Next, by exploring the details of a ρ-bounded algorithm, we provide an improved algorithm with a competitive ratio of at most 16.74.

In the remaining part of this paper, we first present a general framework in Sect. 2. Then we provide an improved online algorithm in Sect. 3. We conclude the paper with open questions in Sect. 4.

2 A general framework

For the online moldable task scheduling problem, let $\bar{s} = \{s_1, s_2, \ldots, s_n\}$ be an allocation of machines to the n jobs, where s_i is the number of machines assigned to the task i. Note that the problem is a rigid task scheduling problem if a fixed $\bar{s}$ is used. Define

$$\mathrm{LB}(\bar{s}) = \max\left\{\frac{1}{m}\sum_{j=1}^{n}(s_j \cdot p_{j,s_j}),\ \max_{j\in\{1,\ldots,n\}}\{p_{j,s_j}\}\right\}.$$

Then $\mathrm{LB}(\bar{s})$ is a lower bound for the makespan of a rigid task scheduling problem. The first item in $\mathrm{LB}(\bar{s})$ refers to the total work load (the product of the number of machines and its processing time) evenly divided by the number of machines, and the second item refers to the length of a task with largest processing time. Clearly, $\mathrm{LB}(\bar{s})$ is lower bounds on the makespan of the optimal offline schedule.

As introduced by Ludwig and Tiwari (1994), a ρ-bounded algorithm is defined as follows.

Definition 1 For any value $\rho \geq 1$, an algorithm for the rigid task scheduling problem is said to be *ρ-bounded* if the makespan generated by this algorithm is at most $\rho LB(\bar{s})$ for any given $\bar{s}$.

The rigid task scheduling problem can be viewed as the strip packing problem, where the width of the strip corresponds to the machines and the height of the strip corresponds to the time axis. The number of machines s and processing time $p_{j,s}$ of a rigid task J_j are corresponding to the width and height of a rectangle item that are requested to be packed in strip packing. Strip packing differs from scheduling rigid tasks in that we have just the additional constraint that a job must be scheduled on consecutive machines. Therefore, algorithms for strip packing can be used for scheduling rigid tasks, but in general not vice versa. Now we know that the known algorithms in Baker and Schwartz (1983), Hurink and Paulus (2007), Ye et al. (2009) can be applied for the online rigid parallel task scheduling problem. Moreover, one can check that only the lower bounds in $\mathrm{LB}(\bar{s})$ are used as lower bounds on the makespan of the optimal offline schedule in Baker and Schwartz (1983), Hurink and Paulus (2007), and Ye et al. (2009). Therefore, algorithms in Baker and Schwartz (1983), Hurink and Paulus (2007), and Ye et al. (2009) are ρ-bounded according to Definition 1, which achieve competitive ratios of 6.99 and 6.66, respectively.

The principle of an offline ρ-dual approximation algorithm (Hochbaum and Shmoys 1987) is to take a positive real number α as input, and either deliver a schedule of length at most $\rho\alpha$ or point out that there exists no schedule of length at most α.

For online problems, it is quite natural to consider greedy algorithms. However, even for rigid task scheduling (Ye et al. 2009), a greedy method that assigns each task to processors such that its completion time is minimized cannot be constant-approximated. In this paper, we will combine techniques used in both offline and online algorithms.

The detailed description of our online algorithm for the target problem is called Algorithm 1 (FRAMEWORK), which is given as follows.

Algorithm 1 is a general framework, in which any ρ-bounded online rigid task scheduling algorithm can be applied as a black box. The key idea of our online algorithm is to adopt the standard doubling technique (e.g., Aspnes et al. 1997) to guess a positive real number α and assign the incoming job in the schedule within the length of $\rho\alpha$ if this is possible; otherwise, we double α until this job can be assigned. To make our idea work, we must guarantee that if the online rigid task scheduling algorithm cannot schedule the incoming jobs within the length of $\rho\alpha$, then no optimal offline algorithm for the non-rigid tasks problem can complete all the jobs within a length of α.

In the following, we discuss the algorithm FRAMEWORK in detail. The algorithm finds an assignment for any incoming job J_j (lines 2–19). We denote the part of the pro-

Algorithm 1: FRAMEWORK (a general framework for the moldable task scheduling problem)

```
Input: A set of $n$ moldable jobs, and a set of $m$ machines. A
       $\rho$-bounded rigid task scheduling algorithm RTA.
1  Let $\alpha = \min_{1\le s\le m}\{p_{1,s}\}$, $T = 0$, $i = 1$, $W_i = 0$, $\alpha_i = \alpha$,
   $bout_i = \emptyset$ ;   /* We denote the part of the
   process during which the value of $\alpha$
   remains unchanged (i.e., $\alpha$ is not updated)
   as a bout. We let the work of a job be the
   product of the processing time and the
   number of machines allocated to this job,
   and $W_i$ be the total work of the jobs
   assigned in the $i$th bout. */
2  foreach incoming job $J_j$ do
3      Let $Is\_scheduled = False$;
4      while $Is\_scheduled = False$ do
5          $f(j) = \{s \mid p_{j,s} \le \alpha_i\}$ ;   /* $f(j)$ is the set
           of machine numbers for job $J_j$ such
           that for each $s \in f(j)$, the processing
           time of job $J_j$ is at most $\alpha_i$ if using
           $s$ machines. */
6          if $f(j) \ne \emptyset$ then
7              Find $s \in f(j)$ such that $s \cdot p_{j,s}$ is minimized;
8              $W_i = W_i + s \cdot p_{j,s}$;
9          if $W_i \le \alpha_i \cdot m$ and $f(j) \ne \emptyset$ then
10             Let $Is\_scheduled = True$;
11             Assign $s$, the number of machines, to job $J_j$;
12             $bout_i = bout_i \bigcup \{J_j\}$;
13             Assign this job $(s, p_{j,s})$ by a rigid task scheduling
               algorithm RTA in the time interval $[T, T + \rho\alpha_i]$;
14         else
15             $T = T + \rho\alpha_i$;
16             $\alpha = 2\alpha_i$;
17             $i = i + 1$;
18             $W_i = 0$;
19             $\alpha_i = \alpha$;
Output: The schedule for each job $J_j$, i.e., an assignment of
        the number of machines assigned to $J_j$ and the start
        time of $J_j$.
```

cess during which the value of α remains unchanged (i.e., α is not updated) as a *bout*, and denote the set of jobs assigned in the ith bout as $bout_i$. Let the *work* of a job be the product of the processing time and the number of machines allocated to this job, and W_i be the total work of the jobs assigned in the ith bout. Let α_i be the value of α in the ith bout. Without loss of generality, we assume that the current bout is i upon the arrival of job J_j. Let $f(j)$ denote the set of numbers of machines for job J_j such that for any number $s \in f(j)$, the processing time of job J_j is at most α_i if using s machines (line 5). If $f(j)$ is not empty, i.e., there exists some number s of machines such that the processing time of job J_j is at most α_i using s machines, then we update the total work of $bout_i$ by adding up the minimum work of job J_j whose processing time is limited to at most α_i (lines 6–8). If $W_i \le \alpha_i \cdot m$ and $f(j) \ne \emptyset$, then lines 9–13 make sure that job J_j can be successfully assigned by a given black-box ρ-bounded algorithm RTA (the detailed proof will be given in Lemma 1), and job J_j is assigned to $bout_i$. Lines 15–19 deal with the case that job J_j cannot be assigned in $bout_i$, which also implies that $bout_i$ is ended and job J_j will be forwarded to the next bout. In $bout_{i+1}$, we let $\alpha_{i+1} = 2\alpha_i$ (line 16 and line 19). The start time of $bout_{i+1}$ is $T + \rho\alpha_i$, where $T = \sum_{1\le k\le i-1} \rho\alpha_k$ (line 15) that can be determined by the iterations. Lemma 2 will show that if job J_j cannot be scheduled in $bout_i$, then any optimal makespan must be larger than α_i. This motivates us to increase the value of α.

Lemma 1 *Suppose the subroutine RTA for the rigid task scheduling problem in Algorithm 1 (FRAMEWORK) is a ρ-bounded algorithm. For any $i \ge 1$, it is feasible to complete all jobs in $bout_i$ within the time interval $[T, T + \rho\alpha_i]$, where $T = \sum_{k=1}^{i-1} \rho\alpha_k$.*

Proof According to Algorithm 1, for any job $J_j \in bout_i$, suppose the number of machines assigned to job J_j is s, then we have $p_{j,s} \le \alpha_i$. Moreover, in each $bout_i$, we have that the total work sum W_i of all jobs in $bout_i$ is no more than $\alpha_i \cdot m$. Suppose the machine assignments for all the jobs in $bout_i$ is $\bar{s}$; then, $\mathrm{LB}(\bar{s}) \le \alpha_i$. Since the subroutine RTA for the rigid task scheduling problem is a ρ-bounded algorithm, all of the $bout_i$ jobs can be completed by a schedule with a length of at most $\rho\mathrm{LB}(\bar{s})$, and thus $\rho \cdot \alpha_i$. Hence, all jobs in $bout_i$ can be scheduled in $[T, T + \rho\alpha_i]$, with $T = \sum_{k=1}^{i-1} \rho\alpha_k$. □

Lemma 2 *For any integer $k \ge 1$ and $bout_k \ne \emptyset$, the makespan of any optimal solution for all the jobs in $bout_{k-1} \bigcup bout_k$ is larger than α_{k-1} (with $\alpha_0 = 0$).*

Proof Let job J_j be the first job in $bout_k$. According to Algorithm 1, either we have $p_{j,s} > \alpha_{k-1}$ for all $s \in \{1, \ldots, m\}$, or there exists an s such that $s \cdot p_{j,s}$ is minimized for $p_{j,s} \le a_{k-1}$ but we have $W_{k-1} + s \cdot p_{j,s} > \alpha_{k-1} \cdot m$. In the former case, we know that the processing time of job J_j is larger than α_{k-1} no matter how many of the m machines are assigned to it. In the latter case, the total work of the jobs in $bout_{k-1} \bigcup \{J_j\}$ is larger than $\alpha_{k-1} \cdot m$. Note that the work of every job in $bout_{k-1} \bigcup \{J_j\}$ is minimized under the assumption that the processing time of that job is at most α_{k-1}. Thus, if any optimal algorithm does not assign a job with a processing time larger than α_{k-1}, then the total work of the jobs in $bout_{k-1} \bigcup \{J_j\}$ must be larger than $\alpha_{k-1} \cdot m$. Hence, the lemma follows immediately. □

Theorem 1 *Given a ρ-bounded rigid task scheduling algorithm as a subroutine, the competitive ratio of Algorithm 1 (FRAMEWORK) for our online moldable task scheduling problem is at most 4ρ. Specifically, there exists a 6.6623-bounded algorithm* (Ye et al. 2009) *for the rigid task scheduling problem, and hence, the competitive ratio of Algorithm 1 is at most 26.65.*

Proof The number of machines for any job is determined upon its arrival, and is not changed later. Moreover, we use the online algorithm RS_r in Ye et al. (2009) as the subroutine RTA that determines the starting time of a job. Thus, one can see that Algorithm 1 is an online algorithm.

Let ALG and OPT be the makespans generated by Algorithm 1 and an optimal offline algorithm, respectively. Upon the arrival of the first job, we let the initial value of α be $\min_s\{p_{1,s}\}$.

The concept of a *bout* was already defined above. Suppose there are totally l bouts until the end of Algorithm 1. Let α_j denote the guessed value of α in the jth bout, i.e., $\alpha_j = 2^{j-1}\min_s\{p_{1,s}\}$.

Lemmas 1 and 2 guarantee that Algorithm 1 either is able to complete successfully all the jobs for the given guessed α or there exists no schedule with a length of at most α. Since there are totally l bouts, we know that the makespan OPT of any optimal algorithm must be strictly larger than α_{l-1}, by Lemma 2.

We have $\alpha_l = 2\alpha_{l-1} = 2^{l-1}\min_s\{p_{1,s}\} < 2OPT$. Let T_j denote the starting time of the jth bout in Algorithm 1 . The rigid task scheduling algorithm is a ρ-bounded algorithm, and Algorithm 1 can complete all the jobs within time $T_l + \rho\alpha_l$ by Lemma 1. Note that $T_j = T_{j-1} + \rho\alpha_{j-1}$ for each $j > 1$ and $T_1 = 0$. Hence, the makespan attained by Algorithm 1 is calculated as follows.

$$\begin{aligned} ALG &= T_l + \rho\alpha_l \\ &= \sum_{j=1}^{l}\rho\alpha_j \\ &\leq 2\rho\alpha_l \\ &\leq 4\rho OPT. \end{aligned}$$

Thus, the competitive ratio of Algorithm 1 is at most 4ρ, and then at most of 26.65. □

3 An improved online algorithm

In this section, we will present an improved algorithm based on the above framework. The algorithm FRAMEWORK reduces a moldable task scheduling problem to a rigid task scheduling problem. In this framework, the rigid task scheduling algorithm is regarded as a black box. Consequently, a natural question arises: Can we take a rigid task scheduling algorithm as a white box and design an improved algorithm for the moldable task scheduling problem? We will show that the answer to this question is affirmative, and give an improved algorithm as described in Algorithm 2 (IOA).

Before we jump into Algorithm 2, let us first give a brief description of the RS_r algorithm in Ye et al. (2009). The algorithm RS_r schedules jobs in shelves. In this algorithm, a job is big if the number of machines s assigned to that job is greater than $m/2$, and small otherwise. Big jobs are never assigned together with other jobs, and each of them is assigned to a separate shelf with height exactly the same as

Algorithm 2: IOA (an improved online algorithm)

```
Input: A set of n moldable jobs, and a set of m machines. A
       ρ-bounded rigid task scheduling algorithm RTA.
1  Let $\alpha = \min_{1\le s\le m}\{p_{1,s}\}$, $T = 0$, $i = 1$, $W_i = 0$, $\alpha_i = \alpha$; Let
   $\beta = 1.56$ and $r = 1.44$; foreach incoming job $J_j$ do
2      Let $Is\_scheduled = False$;
3      while $Is\_scheduled = False$ do
4          $f(j) = \{s \mid p_{j,s} \le \alpha_i\}$ ;   /* $f(j)$ is the set
           of machine numbers for job $J_j$ such
           that for each $s \in f(j)$, the processing
           time of job $J_j$ is at most $\alpha_i$ if using
           $s$ machines. */
5          if $f(j) \neq \emptyset$ then
6              Find $s \in f(j)$ such that $s \cdot p_{j,s}$ is minimized;
7              $W_i = W_i + s \cdot p_{j,s}$;
8          if $W_i \le \alpha_i \cdot m$ and $f(j) \neq \emptyset$ then
9              Let $Is\_scheduled = True$;
10             Assign $s$, the number of machines, to job $J_j$;
11             $bout_i = bout_i \bigcup\{j\}$;
               /* Assign this job $(s, p_{j,s})$ using
               the following method.            */
12             if $s > m/2$ then
13                 Create a shelf of height $p_{j,s}$ at the top of the
                   current schedule, and assign job $J_j$ to this
                   shelf;
14             else
15                 Find an integer $k$ such that
                   $r^{k-1} < p_{j,s} \le r^k$, and then pack $J_j$ into a
                   shelf of height $r^k$ by the FF (First Fit)
                   algorithm. Specifically, the job $(s, p_{j,s})$ is
                   packed leftmost into the lowest shelf of
                   height $r^k$ which has enough room for it.
                   Otherwise, create a new shelf of height $r^k$ at
                   the top of the current schedule, and place this
                   job into it; ;     /* A shelf can be
                   regarded as a rectangle, whose
                   width is $m$ and whose height
                   is determined by the
                   algorithm; the assignment of a
                   shelf is like an operation on
                   a stack, always pushing the
                   current shelf onto the top of
                   the stack. The makespan of the
                   online algorithm is the sum of
                   the heights of all shelves
                   pushed onto the stack. */
16         else
17             $\alpha = \beta \cdot \alpha_i$;
18             $i = i + 1$;
19             $W_i = 0$;
20             $\alpha_i = \alpha$;
Output: The schedule for each job $J_j$, i.e., an assignment of
        the number of machines assigned to $J_j$ and the start
        time of $J_j$.
```

Table 1 An instance: 5 jobs and their processing times on a total of 5 machines

s	J_1	J_2	J_3	J_4	J_5
1	5	2	2	2.5	3.79
2	4	1.5	2	2.5	2.5
3	3	1.1	2	2.1	2.5
4	2	1	2	2	2.5
5	1	1	2	2	2.5

its processing time (line 15 in Algorithm 2). A parameter r is given to distinguish different groups of small jobs. Small jobs are grouped together with similar heights that differ at most r times, and they will be assigned by first fit algorithm among shelves with the same height r^k for some integer k (line 17 in Algorithm 2).

Algorithm IOA is similar to the general framework of Algorithm 1. First, in contrast to Algorithm 1, we use a parameter β in Algorithm 2 (instead of a constant 2 in Algorithm 1) when we update α (line 19 of Algorithm 2). Another difference is that we use the RTA algorithm [i.e., the RS_r algorithm in Ye et al. (2009)] as a white box in Algorithm 2 (lines 14–17). However, the parameter r used in Algorithm 2 may not be the same as that in Ye et al. (2009). Note that the parameter $r = \frac{3}{5-\sqrt{10}} = 1.63$ in Ye et al. (2009). Since Algorithm 1 adopts RTA as a black-box subroutine, the parameter r in Algorithm 1 is also 1.63. Finally, in contrast to Algorithm 1, which always assigns a $bout_i$ job in the time interval $[T, T + \rho\alpha_i]$, with $T = \sum_{k=1}^{i-1} \rho\alpha_k$, Algorithm 2 does not need to maintain the variable T; instead, it will assign the incoming job to the "shelf" that is determined by the algorithm (lines 14–17). A shelf can be viewed geometrically as a rectangle with a width of m and a height determined by the algorithm. A schedule can be viewed as a stack of shelves, with each new shelf laying on top of the last shelf. To illustrate the algorithm IOA, an example is given as follows.

Example 1 We give an example to illustrate how Algorithm 2 works. Let $m = 5$ and $n = 5$. The detailed processing times are given in Table 1. Note that we set $\beta = 1.56$ and $r = 1.44$ in Algorithm 2. After the first job J_1 arrives, $\alpha_1 = 1$, $f(1) = \{5\}$, and $W_1 = 5$. Let s_i be the number of machines assigned by the algorithm IOA to J_i. Then $s_1 = 5$, and $bout_1 = \{J_1\}$. A shelf of height 1 will be created and aligned at the bottom. When the second job J_2 arrives, $f(2) = \{4, 5\}$, $W_1 + 4 * 1 = 9 > 5\alpha_1$, and hence, the first bout is ended. Then it goes to the second bout, i.e., $\alpha_2 = \beta\alpha_1 = 1.56$. In this case, $f(2) = \{2, 3, 4, 5\}$, $W_2 = 3 < \alpha_2 \cdot m = 7.8$. Hence, $s_2 = 2$, and $p_{2,2} = 1.5$. Since $1.44 < 1.5 \le 1.44^2 \approx 2.07$, i.e., $k = 2$, a shelf of height 1.44^2 will be created and assigned onto the top of the first shelf. Upon the arrival of job J_3, $p_{3,s} = 2 > \alpha_2 = 1.56$ for any s, and hence, $f(3) = \emptyset$. In this case, the algorithm goes to bout 3, and $bout_2 = \{J_2\}$. Clearly, $\alpha_3 = \beta\alpha_2 = 1.56^2 = 2.4336$. Then

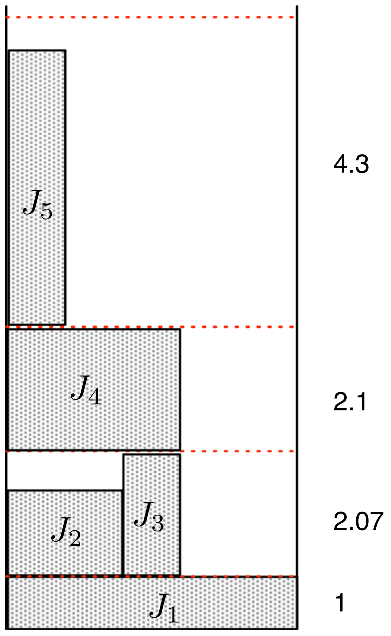

Fig. 1 Illustrating the schedule in Example 1. A shelf is between every two consecutive red dotted line segments, and the numbers on the right are the height of each shelf, respectively

$f(3) = \{1, \ldots, 5\}$, $s_3 = 1$, $p_{3,1} = 2$, and $bout_3 = \{J_3\}$. Since $1.44 < 2 \le 1.44^2$, J_3 can be assigned to a shelf of height 1.44^2, and actually, it is assigned to the right of job J_2. One may check that job J_4 belongs to $bout_3$, and hence, $bout_3 = \{J_3, J_4\}$. Since $s_4 = 3$, we create a shelf of height $p_{4,3} = 2.1$, which will be located on the top of the shelf of height 1.44^2. Up to now, $W_3 = 1 * 2 + 3 * 2.1 = 8.3$. One may also check $f(5) = \emptyset$, and then $\alpha_4 = \beta\alpha_3 = 3.796$. Then $s_5 = 1$, and we find $1.44^3 < 3.79 \le 1.44^4$, which implies that we need to create a shelf of height $1.44^4 \approx 4.3$ on the top of the current schedule, and then assign job J_5 to this shelf. The makespan of this example is 9.47. One could check that the makespan of the optimal solution is at most 3.5 by letting $s_1 = 5, s_2 = s_3 = s_4 = 1, s_5 = 2$ (Fig. 1).

Lemma 3 *Let l be the number of bouts at the end of Algorithm 2 (IOA). The total work sum of the n jobs is upper-bounded by $\frac{\beta^2}{\beta-1} \cdot \alpha_{l-1} \cdot m$, and the longest processing time of any job is upper-bounded by $\beta \cdot \alpha_{l-1}$.*

Proof According to Algorithm 2 (IOA), each incoming job is assigned to one bout. Hence, the union of all the bouts, $\bigcup_{1\le i\le l} bout_i$, consists of all the jobs. Note that if a job J_j is in $bout_i$, then its processing time is upper-bounded by α_i. Thus, the largest processing time of any job is upper-bounded by $\alpha_l = \beta \cdot \alpha_{l-1}$.

Note that the work sum of all the jobs in $bout_i$ is at most $\alpha_i \cdot m$, by Algorithm 2. Hence, the total work of all the n jobs is upper-bounded by $\sum_{i=1}^{l} \alpha_i \cdot m \le \frac{\beta^2}{\beta-1} \cdot \alpha_{l-1} \cdot m$ because $\alpha_i = \beta\alpha_{i-1}$ for any $2 \le i \le l$. □

The final schedule of Algorithm 2 (IOA) consists of a stack of shelves. For simplicity of the proofs, we use the similar notation as in Ye et al. (2009). The shelves created for those jobs to each of which more than $m/2$ machines are assigned are said to be *full*. A shelf of height r^k created for jobs with a degree of parallelism no more than $m/2$ is said to be *sparse*. Otherwise, a shelf is *dense*.

Let H_F denote the total height of all the *full* shelves, H_S denote the total height of all the *sparse* shelves, and H_D denote the total height of all the *dense* shelves, respectively. The following lemma can be obtained directly from Ye et al. (2009).

Lemma 4 (Ye et al. 2009, Theorem 1) *The total work of all the jobs is at least* $H_D \cdot \frac{2}{3r} \cdot m + H_F \cdot m/2$.

Theorem 2 *The online algorithm* IOA *(Algorithm 2) achieves a competitive ratio of at most* 16.74 *by setting* $\beta = 1.56$ *and* $r = 1.44$.

Proof Let ALG and OPT denote the makespans produced by the online algorithm IOA and an optimal offline algorithm, respectively. Let l be the number of bouts at the end of the algorithm IOA. Since $bout_l \neq \emptyset$, we know $OPT \geq \alpha_{l-1}$. By definitions, we have $ALG = H_D + H_F + H_S$.

It is worth noting that there is at most one sparse shelf for each height r^k. Let h be the longest processing time among all the jobs in the sparse shelves, and suppose the shelf to which it is assigned is of a height at most rh. By Lemma 3, we have $h \leq \beta\alpha_{l-1} \leq \beta OPT$. Since each sparse shelf belongs to a different class of height, the value of H_S is at most of the sum of a geometric sequence with the common ratio of r; thus, the following inequality holds.

$$H_S \leq rh + h\sum_{i=0}^{\infty} 1/r^i \leq \frac{r^2}{r-1}h \leq \beta\frac{r^2}{r-1}OPT.$$

Let W be the total work of all the n jobs generated by the online algorithm IOA. By Lemma 4 and letting $r \geq 4/3$, we have

$$\begin{aligned} W &\geq H_D \cdot \frac{2}{3r} \cdot m + H_F \cdot m/2 \\ &\geq \frac{2}{3r}(H_D + H_F)m. \end{aligned}$$

By Lemma 3, we have $W \leq \frac{\beta^2}{\beta-1} \cdot \alpha_{l-1} \cdot m \leq \frac{\beta^2}{\beta-1} \cdot m \cdot OPT$. Clearly, we now have

$$\begin{aligned} ALG &= H_D + H_F + H_S \\ &\leq \frac{3r}{2} \cdot \frac{\beta^2}{\beta-1}OPT + \beta\frac{r^2}{r-1}OPT. \end{aligned}$$

Given any specific number of β, we can find the smallest value of $\frac{3r}{2} \cdot \frac{\beta^2}{\beta-1} + \beta\frac{r^2}{r-1}$. We do a binary search on β in the interval $(1, 2]$. Finally, by letting $\beta = 1.56$ and $r = 1.44$, we have $\frac{3r}{2} \cdot \frac{\beta^2}{\beta-1} + \beta\frac{r^2}{r-1} < 16.74$. Thus, the competitive ratio of the online algorithm IOA is at most 16.74. □

4 Concluding remarks

In this paper, we explored online scheduling of moldable tasks. We presented a constant competitive online algorithm by applying a black-box reduction from the moldable task scheduling to the rigid task scheduling. The techniques used in this paper are based on a combination of the doubling technique and a kind of greedy method. Moreover, we presented an improved algorithm by using a parameter instead of doubling and also taking a known rigid task scheduling algorithm as a white box.

For future work, a natural problem is to design better online algorithms for moldable task scheduling. One may also consider some special cases of the moldable task scheduling problem, for example, the function of processing time is concave or with a monotonic assumption. It would also be interesting to consider malleable task scheduling.

Acknowledgements We would like to thank the anonymous referees for their valuable comments to improve the presentation of this paper. The research of D. Ye was supported in part by NSFC (11671355) and China Scholarship Council. The research of D.Z. Chen was supported in part by NSF under Grants CCF-1217906 and CCF-1617735. The research of G. Zhang was supported in part by NSFC (11271325).

References

Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., & Waarts, O. (1997). On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, *44*(3), 486–504.

Baker, B., & Schwartz, J. (1983). Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, *12*, 508–525.

Belkhale, K., & Banerjee, P. (1990). Approximate algorithms for the partitionable independent task scheduling problem. In *Proceedings of the international conference on parallel processing (ICPP)* (pp. 72–75).

Borodin, A., & El-Yaniv, R. (1998). *Online computation and competitive analysis*. Cambridge: Cambridge University Press.

Cirne, W., & Berman, F. (2001). A model for moldable supercomputer jobs. In *Proceedings of the 15th international parallel and distributed processing symposium (IPDPS)* (pp. 59–66).

Decker, T., Lücking, T., & Monien, B. (2006). A 5/4-approximation algorithm for scheduling identical malleable tasks. *Theoretical Computer Science*, *361*(2), 226–240.

Dutot, P., Mounié, G., & Trystram, D. (2004). Scheduling parallel tasks approximation algorithms, Chapter 26. In J. Y.-T. Leung (Ed.), *Handbook of scheduling: Algorithms, models and performance analysis*. Boca Raton: CRC Press.

Dutton, R., & Mao, W. (2007). Online scheduling of malleable parallel jobs. In *Proceedings of the IASTED international conference on parallel and distributed computing and systems* (pp. 1–6).

Guo, S., & Kang, L. (2010). Online scheduling of malleable parallel jobs with setup times on two identical machines. *European Journal of Operational Research*, *206*(3), 555–561.

Havill, J., & Mao, W. (2008). Competitive online scheduling of perfectly malleable jobs with setup times. *European Journal of Operational Research*, *187*, 1126–1142.

Hochbaum, D., & Shmoys, D. (1987). Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, *34*(1), 144–162.

Hurink, J., & Paulus, J. (2007). Online algorithm for parallel job scheduling and strip packing. In *Proceedings of the 5th international workshop in approximation and online algorithms (WAOA)* (pp. 67–74).

Jansen, K. (2012). A $(3/2 + \varepsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *Proceedings of the 24th ACM symposium on parallelism in algorithms and architectures (SPAA)* (pp. 224–235).

Jansen, K., & Porkolab, L. (2002). Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, *32*(3), 507–520.

Jansen, K., & Thöle, R. (2010). Approximation algorithms for scheduling parallel jobs. *SIAM Journal on Computing*, *39*(8), 3571–3615.

Kalé, L. (2002). The virtualization model of parallel programming: Runtime optimizations and the state of art. In *Proceedings of Los Alamos computer science institute symposium (LACSI)* (pp. 347–364).

Kell, N., & Havill, J. (2015). Improved upper bounds for online malleable job scheduling. *Journal of Scheduling*, *18*(4), 393–410.

Ludwig, W., & Tiwari, P. (1994). Scheduling malleable and nonmalleable parallel tasks. In *Proceedings of the 15th annual ACM-SIAM symposium on discrete algorithms (SODA)* (pp. 167–176).

Mounié, G., Rapine, C., & Trystram, D. (1999). Efficient approximation algorithms for scheduling malleable tasks. In *Proceedings of the 11th annual ACM symposium on parallel algorithms and architectures (SPAA)* (pp. 23–32).

Mounié, G., Rapine, C., & Trystram, D. (2007). A $\frac{3}{2}$-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM Journal on Computing*, *37*, 401–412.

Rapine, C., Scherson, I., & Trystram, D. (1998). On-line scheduling of parallelizable jobs. In *Proceedings of the 4th international europar conference on parallel processing (Euro-Par)* (pp. 322–327).

Saule, E., Bozdağ, D., & Catalyurek, U. (2010). A moldable online scheduling algorithm and its application to parallel short sequence mapping. In *Proceedings of the 15th international conference on job scheduling strategies for parallel processing (JSSPP)* (pp. 93–109).

Turek, J., Wolf, J., & Yu, P. S. (1992). Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the 4th annual ACM symposium on parallel algorithms and architectures (SPAA)* (pp. 323–332).

Ye, D., Han, X., & Zhang, G. (2009). A note on online strip packing. *Journal of Combinatorial Optimization*, *17*(4), 417–423.

Yu, G., Mao, Y., & Xiao, J. (2016). A new lower bound for online strip packing. *European Journal of Operational Research*, *250*(3), 754–759.