Generalized Online Malleable Job Scheduling on Three Processors

Jessen Havill
Department of Math and Computer Science
Denison University
Granville, OH, 43023
havill@denison.edu

Nat Kell
Department of Math and Computer Science
Denison University
Granville, OH, 43023
kell_n3@denison.edu

ABSTRACT

In this paper, we study an online scheduling problem where we consider *malleable jobs*—jobs that can be scheduled to run on any number of processors on a parallel computer. Under the assumptions of our particular model, we show that when m, the number of processors, is three that the competitive ratio of the optimal online algorithm must be between $\frac{1}{8}(5+\sqrt{73}) \approx 1.693$ and 2.123. We also give an algorithm for the model with constant overhead penalty that approaches an asymptotic competitive ratio of $4/3$.

1. INTRODUCTION

Online algorithms process the input of a problem as a sequence of requests. Each piece of input must be handled individually before moving on to the next; hence, the algorithm must make assignments and decisions without knowing information about future requests.

After all requests have been processed, there will most likely be decisions that were not optimal in hindsight. This suboptimal performance is usually measured by a *competitive ratio*, the most common performance metric used when analyzing online algorithms. Formally, let σ denote an input sequence processed by online algorithm A, and let $A(\sigma)$ and $OPT(\sigma)$ denote the cost of σ incurred by A and the optimal solution, respectively. The competitive ratio of A for a minimization problem is defined as

$$C_A = \max_{\sigma} \left\{ \frac{A(\sigma)}{\mathrm{OPT}(\sigma)} \right\} .$$

Informally, C_A is the worst case ratio, over all input sequences, between the cost of the online algorithm and the cost of the optimal. Note that the closer C_A is to 1, the better the algorithm (an online algorithm with $C_A = 1$ always performs optimally).

In our particular problem, we consider online algorithms that distribute the workload of a computer over multiple processors. As the number of processors used in modern computers continues to increase, it is vital to implement scheduling algorithms that distribute a computer's workload among these processors effectively. Since this workload must usually be distributed on the fly (e.g., as a person is actively using a computer), it makes sense to study these scenarios in the context of online algorithms.

There are several different paradigms of online scheduling that have been studied, each with slightly different assumptions and features. In our specific model, we consider scheduling *malleable jobs*, which have the ability to run in parallel on any number of processors, as opposed to non-malleable jobs, which must run in parallel on a fixed number of machines given by the input of the problem.

When a job is scheduled to run on multiple processors, we would expect an overall speedup in its execution time. We can also usually assume that the more machines we utilize when scheduling a job, the greater this speedup will be; however, we must also consider that parallel implementations require extra "bookkeeping" tasks, such as setting up a parallel algorithm, thread synchronization, and communication between machines

Proceedings of the 2011 Midstates Conference on Undergraduate Research in Computer Science and Mathematics

during execution. The time accumulated through maintaining these additional tasks is referred to as *overhead*. As we increase the number of processors we utilize, the greater this overhead time will be.

To capture this tradeoff, we assume that if an online algorithm schedules a job j with processing requirement p_j and overhead penalty c_j to run on k_j processors, the overall execution time of the job becomes $p_j/k_j+(k_j-1)c_j$, where p_j/k_j models the processing speedup and $(k_j-1)c_j$ attempts to capture the added overhead time. Allowing the overhead penalty c_j to vary from job to job models the fact that some tasks will be harder to run in parallel and will require more overhead time. For the algorithm we provide for $m = 2$ (the number of machines), we will use a less general model and assume $c_1 = c_2 = \ldots = c_n$. For the results we provide when $m = 3$, we will use the more general model with job-dependent overhead. This model was first examined in [4], and although alternative models may be considered, this model has recently been empirically validated using standard benchmarks [5].

Formally stated, the input to our problem is a sequence σ of n jobs, where each job j is given by a processing requirement p_j and an overhead penalty c_j. An online algorithm A must sequentially handle the jobs in σ one at a time, assigning each job both a start time $s_j \geq 0$ and a number of processors $k_j \in \{1, 2, \ldots, m\}$ to run on. A can make current assignments based on the previously crafted schedule, but these prior decisions may not be altered. We do not allow A to "backfill", i.e., schedule jobs in areas of idle time that occurred earlier in the schedule; hence, if A's assignment of job j uses machine h, s_j must be greater than the completion time of the last job running on h in the current schedule (before j is scheduled). We do not consider release times or precedence constraints (other than sequential order).

We use competitive analysis to measure the performance of our algorithms, where the objective is the *makespan* of the schedule, or the completion time of the job which finishes last, for our cost function. Formally, let $C_A(\sigma)$ and $C^*(\sigma)$ be the makespan of σ as scheduled by the online algorithm A and the optimal solution, respectively. We say that A is *asymptotically* ω-competitive if for all $\sigma, C_A(\sigma) \leq \omega C^*(\sigma)+t$, where t must be constant relative to σ. If there is no additive term t, we say that the *strong* competitive ratio of A is ω.

In this paper, we prove that any online algorithm for the job-dependent overhead model must have a strong competitive ratio of at least $\frac{1}{8}(5+\sqrt{73}) \approx 1.693$ when $m = 3$. This is an improvement over the trivial lower of bound of $5/3$, which was proved for the constant overhead model in [10] but still applies for the job-dependent overhead model. We also provide an algorithm for the constant overhead model when $m = 2$ that approaches an asymptotic competitive ratio of $4/3$. This algorithm's competitive ratio depends on a parameter ϵ, which is fixed from the beginning of the algorithm. The closer ϵ is set to 0, the closer the asymptotic competitive ratio of the algorithm gets to $4/3$, but the additive term t (discussed earlier) becomes larger, so there is somewhat of a tradeoff. We also note that when $\epsilon = 2$, this algorithm is identical to SET when $m = 2$ [10] and identical to the $3/2$-competitive algorithm in [9] when $\epsilon = 1$. Finally, we give a 2.123-competitive algorithm for the job-dependent overhead model when $m = 3$, thus showing the competitive ratio for the optimal online algorithm must be between 1.693 and 2.123 when $m = 3$.

2. BACKGROUND

Online scheduling was first studied by Graham in 1966 [7]. He considered a simple model, where jobs in a multiprocessor environment must simply be assigned a start time and a single processor to run on. Graham analyzed the basic algorithm *list scheduling*, which assigns jobs to be scheduled as early as possible as soon as they become available for scheduling, and showed its competitive ratio to be $2 - 1/m$ (where, once again, m is the number of processors). Since then, several different models have been considered, as well as further investigation of Graham's "traditional" model. It has been shown for when $m = 2$ and $m = 3$ in this traditional model, *list scheduling* is optimal using competitive analysis [6].

Interestingly, this is not the case when $m = 4$, i.e., there are algorithms for this traditional model that are better than $7/4$-competitive (for instance, the algorithm in [3]). In fact, to the best of our knowledge, it is still an open question as to what is the competitive ratio of the optimal online algorithm for $m = 4$. This points to the fact that even in the most basic schedul-

ing problems, the simple addition of a single processor can add layers of complexity, and in turn, has resulted in research that attempts to improve on already seemingly tight bounds. For instance, the main focus of [3] was to tighten the lower and upper bounds for traditional model when $m = 4$ from 1.7071 and 1.7374 to a slightly improved 1.7310 and 1.7333. Even since then, the lower bound was again raised to $\sqrt{3} \approx 1.7321$ in [2]. We point this out to simply show that even though the results presented in this paper (namely, our bound for the job-dependent overhead model when $m = 3$) might at first seem unnecessarily complex to achieve such a small tightening of bounds, results of this flavor are fairly common in the realm of online scheduling.

Turning our attention to our problem, the studied algorithms for both the constant and non-constant overhead models are fairly straightforward. The model with constant overhead has received more attention due to its simplicity. Havill and Mao examined the basic algorithm *shortest execution time* or SET, which has competitive ratio $4(m-1)/m$ and $4m/(m+1)$ for when m is even and odd, respectively [10]. This is currently the best known algorithm when considering $m \to \infty$ for the constant overhead model. Havill and Mao also provided lower bounds that show any online algorithm for the constant overhead model must have strong competitive ratio at least $3/2$ and $5/3$ for when $m = 2$ and $m = 3$, respectively. Dutton and Mao studied the algorithm *earliest completion time* or ECT, which schedules jobs such that their completion times are as early as possible [4]. They showed that ECT has competitive ratios 2, 9/4, 20/9, for $m = 2, 3$, and 4.

Guo and Kang were the first to consider the model with job-dependent overhead [8]. They gave an algorithm for $m = 2$ that is ϕ-competitive (where $\phi = (1 + \sqrt{5})/2$, or the golden ratio). They also showed that this is the optimal online algorithm for $m = 2$. Most recently, Havill provided a simpler ϕ-competitive algorithm for the model with job-dependent overhead and algorithms that are asymptotically 3/2 and 5/3 competitive for $m = 2$ and 3, respectively, for the constant overhead model [9].

We will finally mention that an alternative model that differs from both our model and the traditional model is multiprocessor scheduling with rejection, or MSR, studied by Bartal et al. in [1]. This model is identical to the traditional model studied by Graham, but with added feature that jobs may be "rejected" at the cost of a job-dependent penalty. Although not entirely identical, one can observe that this seems to resemble our job-dependent overhead model. In fact, the optimal online algorithm for MSR when $m = 2$ is also ϕ-competitive. However, the competitive ratio of the optimal online algorithm for $m = 3$ is still undetermined for MSR, with the range currently being 1.839 to 2. We point this out simply because MSR seems to be a simplified version of our job-dependent overhead model. If this intuition proves to be true, it is unlikely that the bounds for the job-dependent overhead model for $m = 3$ will be easily tightened given the current state of research for MSR.

3. LOWER BOUND

We begin by providing a lower bound on the competitive ratio of any online algorithm for the job-dependent overhead model when $m = 3$.

THEOREM 1. *The strong competitive ratio for any online algorithm for the job-dependent overhead model must be at least* $\omega = \frac{1}{8}(5+\sqrt{73}) \approx 1.693$ *when* $m = 3$.

PROOF. The proof will be framed as a contest between an adversary and an arbitrary algorithm A. We will be begin by defining two recurring types of jobs that will be issued by the adversary the proof, which will be denoted as Type I and Type II. Both job types will be defined by a set of pre-conditions and post-conditions. Throughout the proof, if a job is denoted as a Type I or II job, it can be verified that the pre-conditions defined by the job's respective type are met. We will argue in the definition of each job type why the post-conditions must hold, and so these arguments will be excluded from the body of the proof.

Notationally, let $j_i = (p_i, c_i)$ denote a job j_i with processing requirement p_i and overhead c_i. Let C^j denote the makespan of A's schedule after job j has been scheduled, and let E^j be the earliest time the next job can start execution after j has been scheduled. Let $T[j]$ be the execution time of job j based on how it was assigned by A. We will use B as an arbitrarily large constant. Finally, we note that if a job is denoted by sub- and super-scripts, e.g. j_x^y, then k_x^y will denote how

many processors j_x^y is assigned to.

We are now ready to define Type I and Type II jobs. We first provide general intuition behind the definition of each job type. Type I jobs can be viewed as "filler" jobs with arbitrarily large overhead constants and therefore must be assigned to one processor in order to stay competitive. They are designed to fill the idle time running concurrently with a job assigned to two processors at the end of the current schedule before the Type I job is issued by the adversary. It is important that their processing requirement is the exact same as the idle time they are intended to fill (so from that point on in the proof, we can work with an evenly loaded schedule), but it is even more important that these jobs are long enough so that A will be at least ω-competitive if A chooses not to fill the idle time.

Type II jobs also have arbitrarily large overhead constants, so they too must be assigned to only one processor. A Type II job can viewed as relatively large job with more processing requirement than the makespan of the current schedule before it is issued by the adversary. Thus the optimal schedule will have "saved" a processor for this job to run on from the very beginning of the schedule, only utilizing the other two processors for the rest of the jobs. Meanwhile, A will be forced to schedule the Type II job on a already loaded machine since the adversary will have forced A to use all three processors for the jobs issued before the Type II job. We now provide the formal definitions.

Type I Jobs: We will denote Type I jobs as j_n^I where j_n is the job scheduled immediately before j_n^I. Let $k_1, k_2, \ldots, k_n$ be the current assignments made by A before j_n^I was issued.

Pre-conditions:

1. $k_n = 2$.
2. $E^{j_{n-1}} = C^{j_{n-1}}$, or j_n is the only job A has scheduled thus far.
3. The overhead term of j_n^I is B.
4. Let the processing requirement of j_n^I be p, where $(p + C^{j_n})/C^{j_n} \geq \omega$.
5. $p = T[j_n]$

Post-conditions:

1. A must schedule j_n^I to run concurrently with j_n on one machine. Since j_n^I's overhead term is arbitrarily large, k_n^I must be 1 if A wants to remain ω-competitive. Pre-conditions 2, 4, and 5 ensure that if j_n^I is scheduled after j_n, i.e, on one of the two machines that j_n is running on, then the adversary can stop and A will be at least ω-competitive (see Fig. 1a). Note that we need not consider the makespan of the optimal schedule since if $(p + C^{j_n})/C^{j_n} \geq \omega$, then certainly $(p+C^{j_n})/C^* \geq \omega$, where C^* is the makespan of the optimal schedule.
2. Pre-conditions 2 and 5 ensure that $E^{j_n^I} = C^{j_n^I} = C^{j_n}$, or rather, the makespan of the current schedule does not change after scheduling j_n^I concurrently with j_n, and all three machines are equally loaded (i.e., all three machines have the same completion times).

Type II Jobs: We will denote Type II jobs as $j_{n,g}^{II}$. If the job scheduled immediately before $j_{n,g}^{II}$ is not a Type I job, then j_n was the job scheduled immediately before $j_{n,g}^{II}$. Otherwise, j_n is the job scheduled immediately before the Type I job preceding $j_{n,g}^{II}$. In either case, let $k_n = g$.

Pre-conditions:

1. The overhead term of $j_{n,g}^{II}$ is B.
2. $C^{j_n} = E^{j_n}$. Note that even if j_n is followed by a Type I job, C^{j_n} will still be the earliest start time for $j_{n,g}^{II}$ due to the post-conditions of Type I jobs.
3. Let the processing requirement of $j_{n,g}^{II}$ be p, where $(p + C^{j_n})/C^* \geq \omega$, where C^* is the makespan of the optimal schedule so far.

Post-conditions:

1. The adversary stops after issuing $j_{n,g}^{II}$ and A is at least ω-competitive. Pre-condition 1 ensures that $j_{n,g}^{II}$ must be scheduled on 1 processor. Pre-conditions 2 and 3 together ensure that the best

schedule A can possibly create has makespan $p + C^{j_n}$ and that A will be at least ω-competitive when the adversary stops.

C^* will be created differently based on the schedule before $j_{n,g}^{II}$ is issued. If a Type II job is issued, C^* will always put $j_{n,g}^{II}$ on one processor running from the beginning of the schedule. To denote how the optimal schedule assigns jobs on the remaining two machines, let m_1 and m_2 be the machines to which $j_{n,g}^{II}$ is not assigned, and let M_1^* and M_2^* be the set of optimal job assignments that are processed on m_1 and m_2, respectively (see Fig. 1b). Note that $M_1^* \cap M_2^* \neq \emptyset$ if the optimal algorithm assigns any jobs to two processors, i.e., running on both m_1 and m_2. We also note that the order in which the optimal makes the assignments in M_1^* and M_2^* does not matter.

We now are ready to show how an adversary can force A to be at least ω-competitive. The adversary begins by issuing job $j_1 = (1, c_1)$ where $c_1 = \frac{3-\omega}{6\omega} \approx .128$. If A assigns $k_1 = 1$, then the adversary stops. Since the optimal schedule assigns $k_1 = 3$, A has incurred makespan $1/(1/3 + 2c_1) = \omega$ times the optimal schedule's makespan, and thus A is at least ω-competitive. We now consider the cases where $k_1 = 3$ and $k_1 = 2$ separately.

Case 1, $k_1 = 3$: The adversary continues by issuing $j_2 = (5/2, 1/5)$. We consider the assignment of j_2 in three sub-cases:

Sub-case 1.1, $k_2 = 1$: The adversary stops. Since a better schedule can be created by assigning $k_1 = 3$ and $k_2 = 3$, A has incurred a makespan at least $(17/6 + 2c_1)/(47/30 + 2c_1) \approx 1.6944 > \omega$ times the optimal schedule's makespan, and thus A is at least ω-competitive.

Sub-case 1.2, $k_2 = 2$: The adversary continues by issuing Type I job $j_2^I = (29/20, B)$, and so it follows that A must schedule j_2^I concurrently with j_2 if it wishes to stay ω-competitive. The adversary then finally issues Type II job $j_{2,2}^{II} = (29/10, B)$, $M_1^* = \{k_2 = 2, k_1 = 1\}$, and $M_2^* = \{k_2 = 2, k_2^I = 1\}$, and thus A will also be ω-competitive in this case.

Sub-case 1.3, $k_2 = 3$: The adversary continues by issuing Type II job $j_{2,3}^{II} = (5/2, B)$, where $M_1^* = \{k_1 = 1\}$ and $M_2^* = \{k_2 = 1\}$, and thus A will be ω-competitive in this case.

Case 2, $k_1 = 2$: Case 2 is very similar to case 1, but with sufficient difference that we will argue it explicitly. The adversary continues by issuing Type I job $j_1^I = (c_1 + 1/2, B)$, and thus it follows that A will schedule j_1^I concurrently with j_1 in order to stay ω-competitive. Note that $E^{j_1^I} = 1/2 + c_1$. The adversary continues by issuing $j_3 = (p_3, 2/9)$ where $p_3 = \frac{1}{9}(\frac{25}{2} + \frac{3\sqrt{73}}{2}) \approx 2.812$. We consider the assignment of j_3 in three sub-cases:

Sub-case 2.1, $k_3 = 1$: The adversary stops. Since the optimal algorithm assigns $k_3 = 3$, A has incurred makespan $(1/2 + c_1 + p_3)/(1/2 + c_1 + p_3/3 + 4/9) \approx 1.711 > \omega$ times the optimal schedule's makespan, and thus A will be at least ω-competitive.

Sub-case 2.2, $k_3 = 2$: The adversary continues by issuing Type I job $j_3^I = (p_3/2 + 2/9, B)$, and so it follows that that A must j_3^I must schedule concurrently with j_3 is it wishes to stay ω-competitive. The adversary then finally issues Type II job $j_{3,2}^{II} = (\frac{1}{6}(11 + \sqrt{73}), B)$, $M_1^* = \{k_3 = 2, k_1^I = 1, k_1 = 1\}$, and $M_2^* = \{k_3 = 2, k_3^I = 1\}$, and thus A will also be ω-competitive in this case.

Sub-case 2.3, $k_3 = 3$: The adversary continues by issuing Type II job $j_{3,3}^{II} = (p_3, B)$, $M_1^* = \{k_1 = 1, k_1^I = 1\}$, and $M_2^* = \{k_3 = 1\}$, and thus A will be ω-competitive in this case. □

4. ALGORITHMS

4.1 Constant Overhead

We present an algorithm for the problem with constant overhead on two machines. We use the same definition of $I(j)$ as established in [9], where $I(j)$ is the amount of "blocked-in" idle time (i.e., time where an idle processor sits waiting for the other processor to complete its current job) created by assigning $k_j = 2$. Set $0 < \epsilon \leq 2$. Our algorithm assigns:

$$k_j = \begin{cases} 1 & \text{if } p_j \leq (6 - 2\epsilon)c + (2 - \epsilon)I(j) \\ 2 & \text{otherwise} \end{cases}$$

where s_j is assigned to be as early as possible. Note this is identical to the $3/2$-competitive algorithm in [9]

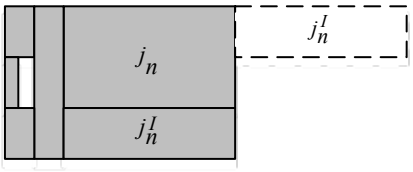

Fig. 1a

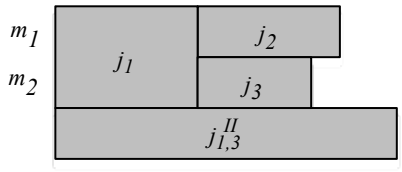

Fig. 1b

Figure 1: **Fig.1a demonstrates how a Type I job must be assigned. Observe that j_n^I takes exactly the same amount of time as j_n running on two machines. The dash-outlined j_n^I shows the schedule configuration that pre-conditions 2, 4, and 5 ensure will be at least ω-competitive. Fig.1b exemplifies a possible optimal configuration after assigning a Type II job. Note how $j_{1,3}^{II}$ runs on one processor from the beginning. In this case, $M_1^* = \{k_1 = 2, k_2 = 1\}$ and $M_2^* = \{k_1 = 2, k_3 = 1\}$.**

if $\epsilon = 1$, and identical to 2-competitive algorithm SET [10] if $\epsilon = 2$.

THEOREM 2. *If $\epsilon < 2$, then the asymptotic competitive ratio of our algorithm for the constant overhead problem when $m = 2$ is $(4-\epsilon)/(3-\epsilon)$. If $\epsilon = 2$, then the strong competitive ratio of our algorithm is 2.*

PROOF. We begin by making an argument symmetric to the proof of Theorem 1 in [9], and likewise, we adapt the same notation. Let t_1^1 denote the time in the online algorithm where a job is executing on one processor concurrently with idle time, and let t_1^2 be time in the online algorithm in which two jobs are both executing on one machine concurrently with each other. Let $J_2 = \{j : k_j = 2\}$ (split on two processors). Denote $\widehat{t_1^1}$ to be the open t_1^1 time at the end of the online algorithms's final schedule, and let $\widetilde{t_1^1} = t_1^1 - \widehat{t_1^1}$. Observe that $\sum_{j \in J_2} I(j) = \widetilde{t_1^1}$. Let C and C^* denote the makespan of the online algorithm and the optimal solution, respectively. We know that

$$\begin{aligned} C^* &\geq \frac{1}{2}\sum_j p_j \\ &= \frac{1}{2}\left(t_1^1 + 2t_1^2 + \sum_{j \in J_2} p_j\right) . \end{aligned} \tag{1}$$

From the online algorithm, we know that

$$\begin{aligned} \sum_{j \in J_2} p_j &> \sum_{j \in J_2} ((6-2\epsilon)c + (2-\epsilon)I(j)) \\ &= (6-2\epsilon)|J_2|c + (2-\epsilon)\widetilde{t_1^1} . \end{aligned} \tag{2}$$

Thus the makespan of the online algorithm's schedule is

$$\begin{aligned} C &= \widetilde{t_1^1} + t_1^2 + \sum_{j \in J_2}\left(\frac{p_j}{2} + c\right) + \widehat{t_1^1} \\ &= \widetilde{t_1^1} + t_1^2 + \frac{1}{2}\sum_{j \in J_2} p_j + |J_2|c + \widehat{t_1^1} \\ &< \left(\frac{4-\epsilon}{6-2\epsilon}\right)\widetilde{t_1^1} + t_1^2 \\ &\quad + \left(\frac{4-\epsilon}{6-2\epsilon}\right)\sum_{j \in J_2} p_j + \widehat{t_1^1} \quad \text{by (2)} \\ &= \left(\frac{4-\epsilon}{6-2\epsilon}\right) t_1^1 + t_1^2 \\ &\quad + \left(\frac{4-\epsilon}{6-2\epsilon}\right)\sum_{j \in J_2} p_j + \left(\frac{2-\epsilon}{6-2\epsilon}\right)\widehat{t_1^1} \\ &\leq \frac{4-\epsilon}{3-\epsilon}\left(\frac{1}{2}t_1^1 + t_1^2 + \frac{1}{2}\sum_{j \in J_2} p_j\right) + \left(\frac{2-\epsilon}{6-2\epsilon}\right)\widehat{t_1^1} \\ &\leq \left(\frac{4-\epsilon}{3-\epsilon}\right) C^* + \left(\frac{2-\epsilon}{6-2\epsilon}\right)\widehat{t_1^1} \quad \text{by (1) .} \end{aligned}$$

We now bound $\widehat{t_1^1}$ given a fixed ϵ using an adversary argument. To create a schedule in which $\widehat{t_1^1}$ is as big as possible, an adversary will first issue j_1 with $p_1 = (6-2\epsilon)c$. The adversary continues to issue jobs with the maximum amount of processing requirement such that each job will be assigned to 1 processor, i.e., if j_n is the nth job issued by the adversary, then $p_n = (6-2\epsilon)c + (2-\epsilon)I(n)$. Let $T(n)$ be the amount of $\widehat{t_1^1}$ time after the algorithm has scheduled the nth job issued by the adversary, or more formally, $T(n) = p_n - I(n)$.

Observe that $T(n-1) = I(n)$ for all $n \geq 2$ and that $T(1) = (6-2\epsilon)c$. Since $T(n)$ depends on $I(n)$, we can define $T(n)$ recursively. Hence, it can be seen that $T(n) = p_n - I(n) = ((6-2\epsilon)c + (2-\epsilon)T(n-1)) - T(n-1) = (1-\epsilon)T(n-1) + (6-2\epsilon)c$. The closed form solution to this recurrence is:

$$T(n) = \frac{2(\epsilon-3)((1-\epsilon)^n - 1)c}{\epsilon}$$

Let $\lim_{n\to\infty} T(n) = R$; hence, R is the upper bound of $\widehat{t_1^1}$. Observe that since $0 < \epsilon \leq 2$, R will be finite. Thus it follows that

$$\begin{aligned} C &\leq \left(\frac{4-\epsilon}{3-\epsilon}\right) C^* + \left(\frac{2-\epsilon}{6-2\epsilon}\right) \widehat{t_1^1} \\ &\leq \left(\frac{4-\epsilon}{3-\epsilon}\right) C^* + \left(\frac{2-\epsilon}{6-2\epsilon}\right) R \end{aligned}$$

We note that although the asymptotic competitive ratio of the algorithm approaches 4/3 as $\epsilon \to 0$, $R \to \infty$ as $\epsilon \to 0$. Also observe that if $\epsilon = 2$, the additive term becomes 0 and the competitive ratio is 2. □

4.2 Job-dependent Overhead

We now present an algorithm for the more general model (i.e., where the overhead term is dependent on each job) when $m = 3$.

Assign s_j to be as early as possible. Let $I_3(j)$ be the longer of the two blocked in "idle times" created by assigning $k_j = 3$. More formally, let $f(i,j)$ be the last completion time of a job on machine i before job j is assigned. Thus $I_3(j) = \max\{s_j - f(i,j) : 1 \leq i \leq 3\}$ given that $k_j = 3$ (see Figure 2). Note that since s_j is always assigned to be as early as possible, at least one element in this set will be 0. Assign:

$$k_j = \begin{cases} 1 & \text{if } p_j \leq \alpha c_j + \beta I_3(j) \\ 3 & \text{otherwise} \end{cases}$$

Where $\alpha \approx 8.763$ and $\beta \approx 1.921$ are the positive solutions to

$$K = 1 - 2\beta/\alpha = 1/3 + 2/\alpha = 4/3(\beta^2/\alpha)\ . \quad (3)$$

Note that although it is legal to assign a job to two processors in this problem, our algorithm never assigns $k_j = 2$.

THEOREM 3. *The competitive ratio of our algorithm for the job-dependent overhead model is no more than* $3 - 4\beta/\alpha = 2K + 1 \approx 2.123$ *when* $m = 3$.

PROOF. Our argument will be structured much like the proof of Theorem 2 in [9], and again, we adapt the same notation. Let t_1^1 denote the time in the online algorithm where a job is executing on one processor concurrently with idle time, and let t_1^2 and t_1^3 be time in the online algorithm in which only two and three jobs are executing on one machine concurrently with each other, respectively. Let $J_3 = \{j : k_j = 3\}$ (split on three processors). Denote $\widehat{t_1^1}$ and $\widetilde{t_1^1}$ to be the same as in the proof of Theorem 2. Observe that $\sum_{j\in J_3} I_3(j) \geq \widetilde{t_1^1}$.

It will be useful to establish certain inequalities based on (3). It can be verified that

$$1.685 \approx 3K < 2K + 1 \quad (4)$$

$$1.921 \approx \alpha/2(1-K) < 2K + 1 \quad (5)$$

$$.421 \approx \beta/2(1-K) < K \approx .562 \quad (6)$$

$$2.562 \approx 2\alpha/3(1-K) < 4K + 2 \approx 4.246 \quad (7)$$

From the online algorithm we know that

$$\sum_{j\in J_3} c_j < 1/\alpha \sum_{j\in J_3} p_j - \beta/\alpha \widetilde{t_1^1}\ . \quad (8)$$

We now consider the makespan of the online algorithm

$$\begin{aligned} C &= \widetilde{t_1^1} + t_1^2 + t_1^3 + \sum_{j\in J_3} \left(\frac{p_j}{3} + 2c_j\right) + \widehat{t_1^1} \\ &= \widetilde{t_1^1} + t_1^2 + t_1^3 + \frac{1}{3}\sum_{j\in J_3} p_j + 2\sum_{j\in J_3} c_j + \widehat{t_1^1} \\ &< (1 - 2(\beta/\alpha))\widetilde{t_1^1} + t_1^2 + t_1^3 \\ &\quad + (1/3 + 2/\alpha)\sum_{j\in J_3} p_j + \widehat{t_1^1} \qquad \text{by (8)} \\ &= K\widetilde{t_1^1} + t_1^2 + t_1^3 + K\sum_{j\in J_3} p_j + \widehat{t_1^1} \\ &= Kt_1^1 + t_1^2 + t_1^3 + K\sum_{j\in J_3} p_j + (1-K)\widehat{t_1^1} \\ &= 3K\left(\frac{1}{3}t_1^1 + \frac{2}{3}t_1^2 + t_1^3 + \frac{1}{3}\sum_{j\in J_3} p_j\right) \\ &\quad + (1-K)\widehat{t_1^1} - (3K-1)t_1^3 - (2K-1)t_1^2\ . \end{aligned}$$

We now proceed by cases based on the existence of $\widehat{t_1^1}$ and how the optimal solution assigns the job that finishes last in the algorithm's schedule. For the first

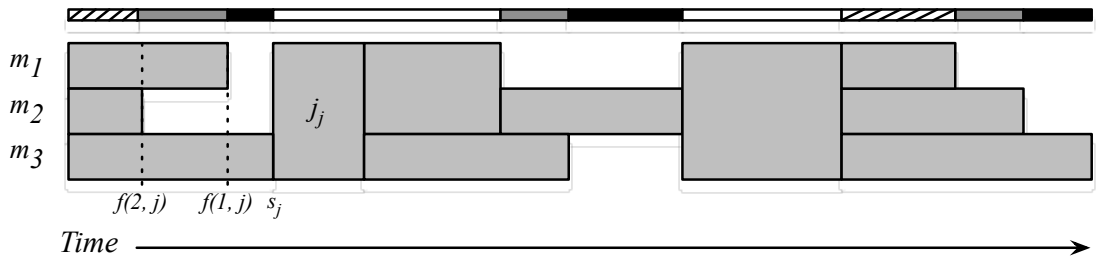

Figure 2: **Example of possible schedule created by the algorithm for when $m = 3$, illustrating the definitions in both Theorems 2 and 3. The black, gray, and striped areas (marked by the key above the schedule) denote areas that contribute to the t_1^1, t_1^2, and t_1^3 time, respectively. Note the last black area of the schedule is $\widehat{t_1^1}$. Also observe the illustration of $f(i, j)$ at the beginning of the schedule. In this case, $I_3(j) = s_j - f(2, j)$.**

two cases, we use the fact that

$$C^* \geq \frac{1}{3}\sum_j p_j = \frac{1}{3}\left(t_1^1 + 2t_1^2 + 3t_1^3 + \sum_{j \in J_3} p_j\right) .$$

In the first case, assume that $\widehat{t_1^1} = 0$, or that there is no open idle time at the end of the schedule in which only one machine is running. Thus we know that:

$$\begin{aligned} C &\leq 3KC^* + (1-K)\widehat{t_1^1} - (3K-1)t_1^3 - (2K-1)t_1^2 \\ &\leq 3KC^* < (2K+1)C^* \quad \text{by (4)} \end{aligned}$$

Note that the second inequality holds since $2K-1 > 0$, and t_1^3 and t_1^2 are both non-negative.

For the rest of the cases, assume that $\widehat{t_1^1} > 0$. Let job ℓ be the job in the algorithm that finishes last. Observe that since $\widehat{t_1^1} > 0$, $k_\ell = 1$ and there is time at the end of the schedule where job ℓ is running by itself. For the second case, assume that in the optimal schedule ℓ is assigned to 1 processor. It follows that $C^* \geq p_\ell$. Thus we can say that:

$$\begin{aligned} C &\leq 3KC^* + (1-K)\widehat{t_1^1} - (3K-1)t_1^3 - (2K-1)t_1^2 \\ &\leq 3KC^* + (1-K)p_\ell \leq (2K+1)C^* . \end{aligned}$$

For the third case, assume that the optimal schedule assigns ℓ to 2 processors. Hence we know that

$$C^* \geq \frac{1}{3}\left(t_1^1 + 2t_1^2 + 3t_1^3 + \sum_{j \in J_3} p_j\right) + c_\ell . \qquad (9)$$

Therefore

$$\begin{aligned} C &\leq 3K(C^* - c_\ell) + (1-K)\widehat{t_1^1} \\ &\quad - (3K-1)t_1^3 - (2K-1)t_1^2 \quad \text{by (9)} \\ &\leq 3K(C^* - c_\ell) + (1-K)\widehat{t_1^1} \\ &\quad - (2K-1)I(\ell) \quad \text{since } t_1^2 + t_1^3 \geq I(\ell) \\ &= 3K(C^* - c_\ell) + (1-K)(p_\ell - I(\ell)) \\ &\quad - (2K-1)I(\ell) \\ &\leq 3K(C^* - c_\ell) - (2K-1)I(\ell) \\ &\quad + (1-K)(\frac{p_\ell}{2} - c_\ell + C^* - I(\ell)) \\ &\quad \text{since } C^* \geq p_\ell/2 + c_\ell \\ &= (2K+1)C^* - (2K+1)c_\ell - KI(\ell) \\ &\quad + (1-K)\frac{p_\ell}{2} \\ &\leq (2K+1)C^* - (2K+1)c_\ell - KI(\ell) \\ &\quad + (1-K)\frac{\alpha c_\ell + \beta I(\ell)}{2} \\ &= (2K+1)C^* + (\alpha/2(1-K) - (2K+1))c_\ell \\ &\quad + (\beta/2(1-K) - K)I(\ell) \\ &\leq (2K+1)C^* \quad \text{by (5) and (6) .} \end{aligned}$$

In the final case, assume that ℓ is assigned to 3 processors in the optimal schedule. Thus we know that

$$C^* \geq \frac{1}{3}\left(t_1^1 + 2t_1^2 + 3t_1^3 + \sum_{j \in J_3} p_j\right) + 2c_\ell . \qquad (10)$$

Through a similar argument made in case 3, it can be

seen that

$$
\begin{aligned}
C &\leq 3K(C^* - 2c_\ell) - (2K-1)I(\ell) \\
&\quad + (1-K)(\frac{2}{3}p_\ell - 2c_\ell + C^* - I(\ell)) \\
&\quad \text{since } C^* \geq p_\ell/3 + 2c_\ell \\
&= (2K+1)C^* - (4K+2)c_\ell - KI(\ell) \\
&\quad + (1-K)\frac{2}{3}p_\ell \\
&\leq (2K+1)C^* - (4K+2)c_\ell - KI(\ell) \\
&\quad + (1-K)\frac{2(\alpha c_\ell + \beta I(\ell))}{3} \\
&= (2K+1)C^* + (2\alpha/3(1-K) - (4K+2))c_\ell \\
&\quad + (4\beta/3(1-K) - K)I(\ell) \\
&= (2K+1)C^* + (2\alpha/3(1-K) - (4K+2))c_\ell \\
&\quad + (K-K)I(\ell) \quad \text{by (3)} \\
&\leq (2K+1)C^* \quad \text{by (7)} .
\end{aligned}
$$

□

5. CONCLUSION

We have shown that for the more general model with job-dependent overhead, any online algorithm must be at least $\frac{1}{8}(5+\sqrt{73}) \approx 1.693$-competitive when $m = 3$. Furthermore, we have given an algorithm for this same model that we have shown is no more than 2.123-competitive. Thus the optimal online algorithm when $m = 3$ must be between these two bounds. We also have given an algorithm for the model with constant overhead penalty when $m = 2$ that approaches an asymptotic competitive ratio of $4/3$. This algorithm also connects the 3/2-competitive algorithm presented in [9] and the 2-competitive algorithm SET studied in [10], showing that when $m = 2$, these algorithms only differ based on what we initially pick for ϵ.

There are several different open questions to consider for future research:

- What is the competitive ratio of the optimal online algorithm for the job-dependent overhead model when $m = 3$? Given that this question has not been answered for $m = 3$ for the model studied in [1], it is unlikely that this answer will be easily obtained for our model; however, as to how much we can tighten this range is a question that deserves more attention.

- What is the lower bound on the asymptotic competitive ratio for the constant overhead model for when $m = 2$, or more specifically, can one do better asymptotically than our algorithm? (Note that the lower bound in [10] for $m = 2$ only applies to strong competitive ratios).

- The algorithms in [9] are asymptotically competitive. Can one construct algorithms that match the strong competitive ratio lower bounds in [10]?

- The basic algorithm SET has competitive ratio 4 as $m \to \infty$ [10] . Can one construct a better algorithm which approaches a lower competitive ratio as $m \to \infty$?

6. ACKNOWLEDGEMENTS

We would like to thank the Anderson family for their generous funding that made this project possible.

7. REFERENCES

[1] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, L. Stougie. Multiprocessor scheduling with rejection. *SIAM J. Discrete Math*, 13:64-78, 2000.

[2] R. Chandrasekaran and John Rudin III. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32:717-735, 2003.

[3] B. Chen, A. van Vliet, and G. J. Woeginger. New lower and upper bounds for on-line scheduling, *Oper. Res. Lett.* 16:221-230, 1994.

[4] R. A. Dutton and W. Mao. Online scheduling of malleable parallel jobs. In *Proceedings of the ISATED international Conference on Parallel and Distributed Computing and Systems*, 1-6, 2007.

[5] R. A. Dutton, W. Mao, J. Chen, and I. W. Watson. Parallel job scheduling with overhead: A benchmark study. In *Proceedings of the IEEE International Conference on Networks, Architecture, and Storage (NAS)*, 326–333, 2008.

[6] U. Faigle, W. Kern, and G. Turan. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107-119, 1989.

[7] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical J.*, 45:1563-1581, 1966.

[8] S. Guo and L. Kang. Online scheduling of malleable parallel jobs on two identical machines. *European Journal of Operational Research*, 206:555–561, 2010.

[9] J. T. Havill. Online malleable job scheduling for $m \leq 3$. *Information Processing Letters*, 111:31–35, 2010.

[10] J. T. Havill and W. Mao. Competitive online scheduling of perfectly malleable jobs with setup times. *European Journal of Operational Research*, 187:1126–1142, 2008.