Asymptotic Algorithms for Online Moldable Job Scheduling

Nathaniel Kell, Loc Tran, Evan Lang, Khoi Le

November 2021

1 Introduction

In recent years, the use of data center and cloud services is on the rise. There are many data centers providers like Amazon Web Services or Google Cloud that responsible for the needs of sending and receiving data from users to services providers. Companies like Facebook, Netflix, Spotify, etc are using these data centers as a bridge between them and normal users like us. To help processing coming in data, these organizations must design efficient scheduling algorithms that distribute these workloads to make it possible to complete jobs in an efficient time. Since the data flow is in real time, what the next job coming in will be, how long it will take, or how many more jobs there are, these scheduling algorithms must be online algorithms. Furthermore, with the rising availability and popularity of multi-core computing, it is possible for data centers to spread the workload of any moldable job throughout multiple machines, resulting in more efficient runtime.

1.1 Problem Definition

Given this motivation, we consider the following problem definition: Given n jobs and each job j with processing time p_j, we need to find the optimal way to distribute these jobs among m machines such that the time for the last machine to finish its assigned jobs, or the makespan of the schedule, is minimized. This problem is an online problem. Many researchers have proposed different approaches to this problem, one of which is that we would allow many machines to process a single moldable job. In this research, we define the execution time of a moldable job j with processing time of p_j assigned to k_j machines to be

$$\frac{p_j}{k_j} + (k_j - 1)c$$

where the term $\frac{p_j}{k_j}$ can be interpreted as modeling the speedup as we divide p_j evenly over k_j machines. On the other hand, the term $(k_j - 1)c$ can be interpreted as the added overhead time, or the penalty of dividing the jobs, as in real-life, we have many factors like processor management, job-to-job communication and shared memory accesses that affect the total processing time.

In order to assess the performance of an on-line scheduling algorithm, we will compare the makespan that the algorithm produces with the optimal makespan. Specifically, for a specific on-line scheduling algorithm A with a job sequence R, its performance is measured by comparing its makespan (denoted $A(R)$) with the optimal makespan (denoted $OPT(R)$).

The algorithm A is called *asymptotically α- competitive* if for all job sequence R with T is a constant with respect to R, we have the following:

$$A(R) \leq \alpha \cdot OPT(R) + T \tag{1}$$

In addition, we call the algorithm A strongly α-competitive if for all job sequence R:

$$A(R) \leq \alpha \cdot OPT(R) \tag{2}$$

Intuitively, the smaller the value of α is, the better the algorithm A performs.

1.2 Related Work.

Online Scheduling was first studied by Graham, R. L. in 1969 [1]. He studied a simple model, where the algorithm takes an arbitrary list of jobs as its input and tries to schedule those job on m identical machines. He studied a simple deterministic greedy algorithm, now commonly called List Scheduling. This algorithm schedules the current job on the least loaded machine. The competitive ratio of List Scheduling is $2-\frac{1}{m}$. We will examine a variation of this problem with the ability to parallel processing jobs.

There are numerous publications related to online problem using overhead constant c as Constant Overhead on m Machines (CO_m). It's proved by Havill and Mao [4] in 2008 that the online algorithm SET (Shortest Execution Time) that assigns each job to the number of processors that minimizes that job's individual execution time has competitive ratio $4(\frac{m-1}{m})$ and $\frac{4m}{m+1}$ for all even and odd m, respectively. In 2010, Havill [3] gave an online algorithms for scheduling malleable parallel jobs on m = 2 and 3 machines that are asymptotically $\frac{3}{2}$ and $\frac{5}{3}$ competitive, respectively

In 2015, Kell and Havill [5] proposed an online algorithm OCS for CO_2 and showed that it is strongly $\frac{3}{2}$ competitive, which is an improvement over the ϕ-competitive algorithms in Guo and Kang [2] and Havill 2010 [3] designed for NCO_2. The latest bounds are shown below:

In 2010, Guo and Kang [2] gave a more general perspective of the online malleable job scheduling problem, replacing constant c with job-dependent overhead term c_j. This transform the problem to online problem using job-dependent overhead c_j as Non-Constant Overhead on m Machines (NCO_m), where m may or may not be fixed. It's proved by them that the competitive ratio of this algorithm is equal to $\frac{1+\sqrt{5}}{2}$ - the golden ratio. Since NCO is a more general approach to CO, this competitive ratio is also applies to CO algorithm.

1.3 Our Results

Our result has been focusing on the bounds for the asymptotic case for two machines. In particular, we came up with an algorithm that is $\frac{5}{4}+\varepsilon$-competitive.

Theorem 1. *There exists an online algorithm for CO_2 with asymptotic competitive ratio of $5/4+\varepsilon$.*

Regarding the lower bound for such case, we came up with a sequence of jobs such that for any algorithm, the best competitive ratio it can achieve is $\frac{7}{6}$.

Theorem 2. *The asymptotic competitive ratio of any online algorithm for for CO_2 jobs must be at least 7/6.*

The above sequence also implies a lower bound of $\frac{4}{3}$ on the strong case for two machines, i.e. if we apply the same sequence to the classic case, then the best competitive ratio that any algorithm can achieve is $\frac{4}{3}$.

Theorem 3. *The asymptotic competitive ratio of any online algorithm for for classic identical machine scheduling must be at least 4/3.*

	$m=2$		$m\to\infty$	
Case	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Strong	3/2, shown [4]	3/2, shown in [5]	1.837, shown in [6]	4, shown in [4]
Asymptotic	$?\to 7/6$	$4/3+\varepsilon\to 5/4+\varepsilon$	(?) $e/(e-1)$ (Section 3.2)	4

2 Moldable Job Scheduling

2.1 Improved Asymptotically Competitive Algorithm for CO_2

Proof of Theorem 1. To show the algorithm gives a $\frac{5}{4}$ approximation, we will show that it always has a $\frac{5}{4}$ upper bound in both phases separately, which then makes the algorithm $\frac{5}{4}$ in total.

Algorithm 1: Asymptotically Competitive Algorithm for Moldable Job Scheduling when $m = 2$

```
Define B ← c/ε
Initialize phase of algorithm to be P ← phase1
for job j = 1,2,...,n in online sequence do
    if P = phase1 then
        if I(j) ≤ B then
            if p_j ≤ 3B then
                set k_j ← 1 and assign j to machine 1
            else
                set k_j ← 2
        else
            P ← phase2
            B ← I(j)
    if P = phase2 then
        if p_j ≤ 2B + I(j) then
            set k_j ← 1 and assign j to less loaded machine
        else
            k_j ← 2
            B ← c/ε
            P ← phase1
```

Phase 1:
Consider the final job in phase 1 with a size p. Based on how the first phase works, the size of a block in phase 1 will always consist of some idle time I, followed by the time left by our final job $\frac{p}{2} + c$. This tells us that our total sum of jobs is additionally $I + p$, therefore our optimal solution can be defined as:

$$OPT \geq \frac{I+p}{2}$$

We can also note that, as defined by the algorithm, that $I < B$ and $p > 3B$. This tells us that $p > 3I$, which then gives us $OPT \geq \frac{I+p}{2} = \frac{I+3I}{2} = 2I$, so:

$$OPT \geq 2I$$

For a final note, we can observe that, since $OPT \geq \frac{I+p}{2}$, that $OPT \geq \frac{p}{2} \geq \frac{3B}{2}$. This then further simplifies to give us $OPT \geq \frac{3}{2}(\frac{c}{\varepsilon}) \geq \frac{c}{\varepsilon}$. This tells us our final fact:

$$\varepsilon * OPT \geq c$$

We can now consider the value that our algorithm gives us. As phase 1 has us only assign to one machine, unless we are splitting, we have a total size of $ALG = I + \frac{p}{2} + c$. Using our earlier facts, this tells us the following:

$$ALG = I + \frac{p}{2} + c$$

$$\geq OPT + \frac{I}{2} + c$$

$$\geq OPT + \frac{1}{4}OPT + c$$

$$\geq \frac{5}{4}OPT + \varepsilon OPT$$

$$= (\frac{5}{4} + \varepsilon OPT)$$

Therefore phase 1 of our algorithm gives us a $\frac{5}{4}$ approximation.

Phase 2:
For phase 2, we have to account for 3 separate parts of a block. We denote t_1 and t_2 to be the time in a block during which only one job is executing on one processor and the time during which two jobs are executing concurrently, respectively. In addition, we let P to be the size of the job getting split at the end of phase 2. Based on this, a single block's makespan ($ALGO$) in phase 2 is:

$$ALGO = t_1 + t_2 + \frac{P}{2} + c$$

Given the above parts of the block, we can develop 3 constraints on the optimal makespan (OPT) of a block:

1. The optimal makespan of a block must be at least the average volume of every job in a block, i.e.

$$OPT \geq \frac{1}{2}(t_1 + 2t_2 + P) \tag{3}$$

2. The optimal makespan of a block must be at least the size of the splitted job. Thus,

$$OPT \geq \frac{P}{2}$$

Since $P = t_j + 2B'$, $OPT \geq \frac{t_j + 2B'}{2} \geq \frac{2B'}{2} = B'$. Because of the way we defined phase 2 coming after phase 1, the constant B' must be at least B. In other words, $OPT \geq B = \frac{c}{\varepsilon}$. Multiply both side by ε, we will have:

$$c \leq \varepsilon OPT \tag{4}$$

3. Since in the smallest case, there exists t_1 and P in a block, we will have

$$OPT \geq \frac{t_1 + P}{2}$$

Notice that having t_2 on the right hand side only benefits us because the jobs that are processing in t_2 is included in OPT. By the definition of the algorithm of phase 2, we know that $P \geq t_j + 2B'$, and in this case, $t_j = t_1$. Therefore, we will have

$$OPT \geq \frac{t_1 + t_1 + 2B'}{2} = t_1 + 2B'$$

By the way we define phase 2, the maximum idle time is $2B'$, i.e. $t_1 = t_j \leq 2B'$. Thus, this gives us

$$OPT \geq t_1 + 2B' \geq 2t_1$$

Therefore,

$$t_1 \leq \frac{OPT}{2} \tag{5}$$

Going back to our analysis of $ALGO$, we will have

$$ALGO = t_1 + t_2 + \frac{P}{2} + c$$

$$ALGO = \frac{1}{2}(t_1 + 2t_2 + P) + \frac{t_1}{2} + c$$

By inequalities (4), (5) and (6), we will have

$$ALGO \leq OPT + \frac{OPT}{4} + \varepsilon OPT = (\frac{5}{4} + \varepsilon)OPT$$

This shows that each block of phase 2 is $(\frac{5}{4} + \varepsilon)$-approximation. Therefore, phase 2 of our algorithm also gives us a $(\frac{5}{4} + \varepsilon)$-approximation. Hence, our algorithm is $(\frac{5}{4} + \varepsilon)$-approximation.

□ □

2.2 Asymptotic Lower Bound for Two Machines for Moldable Jobs

In this section we prove Theorem 2. In particular, we create a sequence of jobs for two machines such that when it is applied to any algorithm, the best competitive ratio it can reach is $\frac{7}{6}$, which gives us a lower bound of $\frac{7}{6}$-competitive ratio for the moldable job scheduling for two machines.

Proof of Theorem 2. The proof will be given as a game between any algorithm, and an adversary, who will give jobs for the algorithm to complete such that the approximation given by the algorithm is as bad is possible.

Let a *block of jobs* be the set of jobs that come after a job that is split into two machines, until there is a job that is split again, including the latter split job. We will show that in any given block, the adversary can force at minimum a $\frac{7}{6}$ approximation, and would therefore show a $\frac{7}{6}$ approximation overall.

In our general job setting, the adversary will first give the algorithm a job of a processing time ε, defined to be much smaller than our constant overhead c. While the algorithm does not split the current job, and assigns to the less used machine, the adversary will respond by keeping a 2-1 ratio between the machines, which can be done by assigning a job of a size equal to the sum of the previous jobs in the schedule. This will in turn, give us a 2-1 ratio, which tells us that the size of the load in our smaller machine T is equal to the total idle time created by the larger machine I.

If the algorithm instead assigns a job to the opposite machine, we can see that we will be worse then the 2-1 ratio temporarily. The adversary will then respond to the algorithm making this decision by repeatedly assigning jobs of size ε, until our machines are back to an approximately 2-1 ratio again.

Once the algorithm finally decides to split a job, we have to look at 2 seperate cases:

1. If the algorithm decides to split the first or second job that the adversary assigned it in the block, then since the adversary first assigned jobs of size ε, we must have a total job size of the block be $\frac{\varepsilon}{2}+c$ in the 1 job case or $2\varepsilon+c$ in the two block case. As ε is much smaller than c, we have it so each of these blocks must be much worse then a $\frac{7}{6}$ approximation, since the constant factor severely outweighs having the jobs be assigned to singular machines.

2. If the algorithm decides to split on the third job, or any job afterwards, then we need to examine the best possible approximation that we could get with the jobs our adversary assigns. By the method that the adversary is assigning jobs to the algorithm, we must be in a case where the adversary is currently keeping a 2-1 ratio or worse. When this is the case, the optimal schedule could have the largest job put onto 1 machine, while every other job was put on the other. This tells us that the optimal makespan for the jobs the adversary assigns is half the total job size. because of the 2-1 ratio that the adversary kept then, if we have T be the size of the smaller machine before the final job p_j is loaded, and I be the idle time before p_j is loaded, then we have a optimal makespan of $OPT = p_j = T+T+I = 3I$. The total makespan given by the algorithm, on the other hand, will be the value of the larger machine, plus the time given by the final job p_j. This gives us an overall makespan of $2I+\frac{p_j}{2}+c$.

   If we then find the ratio between the optimal solution, and the solution our algorithm gave us, we are left with:

$$\frac{ALG}{OPT} = \frac{2I+\frac{p_j}{2}+c}{3I} = \frac{2}{3}+\frac{\frac{p_j}{2}}{3I}+\frac{c}{\frac{3}{2}I} = \frac{7}{6}+\frac{c}{\frac{3}{2}I}$$

   Note that as the algorithm goes longer and longer without splitting, that the value of the idle time will increase. This means that since I is increasing, and c is a constant, that $\frac{c}{\frac{3}{2}I}$ approaches 0. This then gives us an approximation of $\frac{7}{6}+0=\frac{7}{6}$.

In both cases, we must have it so we cannot do better than a $\frac{7}{6}$ approximation. Since this is the case for every individual block, then this must work for the entire set of jobs. Therefore there is an asymptotic lower bound of $\frac{7}{6}$ for 2 machines using moldable jobs. □ □

3 Results for Classic Asymptotic Load Balancing

3.1 Asymptotic Lower Bound for Two Machines in Classic Setting

Proof of Theorem 3. In the classic jobs setting, we will be analyzing the same idea as the bound on the makespan scheduling on identical machines, with moldable jobs. This will be done by using an adversary to assign jobs in response to the actions of any given algorithm.

For this instance, we will consider an adversary that behaves identically to the one used in the moldable job scheduling. The only difference to consider, then, is how to analyze the ratio when none of the jobs can be split up.

We can consider the analysis of the classic setting to be like our moldable setting, where instead of looking at blocks, we are instead thinking of the entire schedule as one, where the final job is never split.

In a potential optimal solution then, we would have 0 idle time between the machines, as we could have put the biggest job on 1 machine, and every other job on the other. This gives us an optimal solution of $T + \frac{1}{2}I = \frac{3}{2}I$.

For our algorithm solution, We take the sum between the load of our smaller machine T, and our idle time I. As we have kept a 2-1 ratio, we know these values must be the same, and so we have an algorithmic scheduling of $I + T = 2I$.

If we then solve for the ratio between our algorithm and an optimal solution, we get:

$$\frac{ALG}{OPT} = \frac{2I}{\frac{3}{2}I} = \frac{4}{3}$$

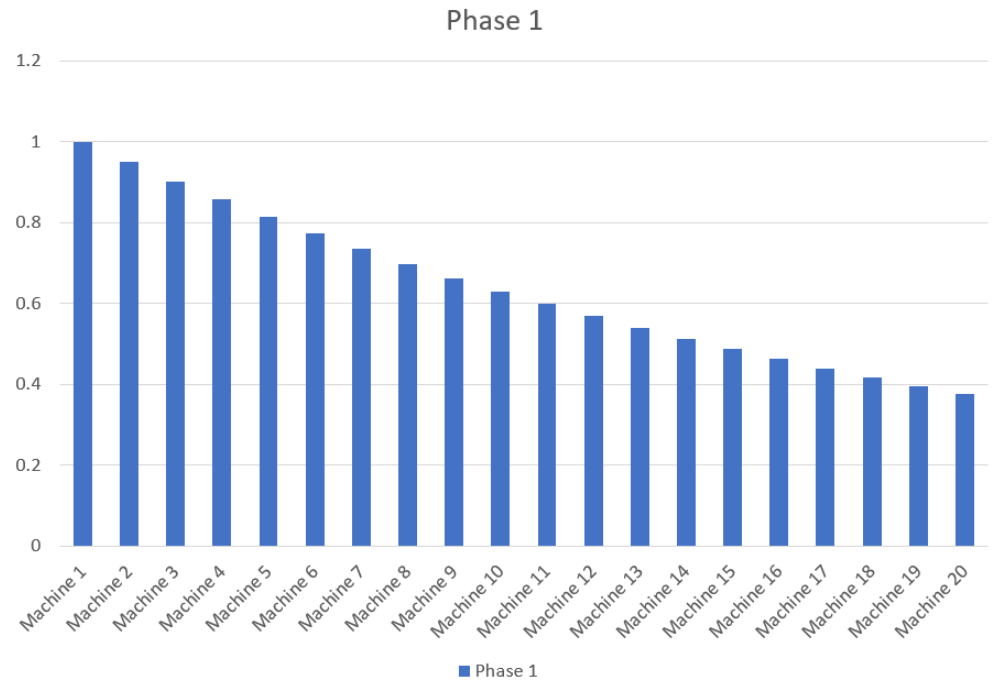
This tells us that even in a best case scenario, we still get an approximation of $\frac{4}{3}$. Therefore in the classic setting, there is an asymptotic lower bound of $\frac{4}{3}$ when we are using 2 machines. □ □

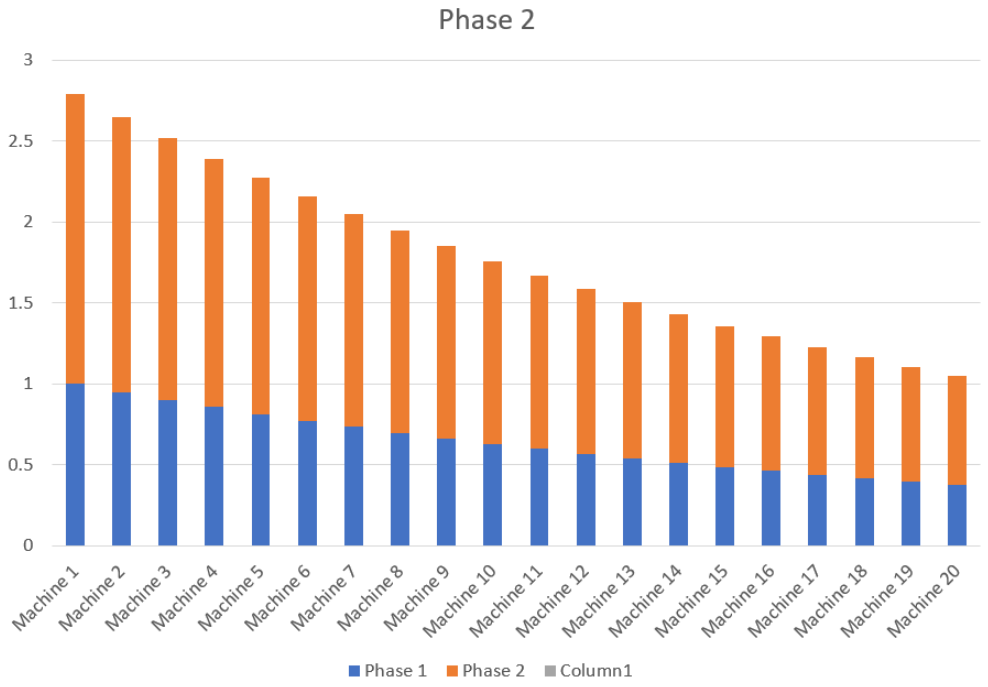
3.2 Experiments for m Machine Lower Bound

We have been experimenting on finding lower bound for the classic scheduling problem. Based on our stimulation on Python, we have a conjecture that the lower bound for the classic scheduling problem converges to $\frac{e}{e-1}$.

The way we set up the lower bound is that we let the number of machines is m, labeled 1 to m. For each of these machines with label j, we assign a job with size $\frac{m-1}{m}^{j-1}$. For instance, machine 1 will first receive a job of size 1 and machine 2 will receive a job of size $\frac{m-1}{m}$. Intuitively, after the assignment of these m jobs, the load of machines from 1 to m is in descending order, and the load of the $(i^+1)th$ machine is equal to $\frac{m-1}{m}$ the load of the i^{th} machine.

In the next steps, we keeps the order of load so that it is descending from machine 1 to machine m, and the ratio between two consecutive machines remains the same ($\frac{m-1}{m}$). For the scheduling of future jobs, we follow the principle of the List Scheduling Algorithm, i.e. assigns jobs greedily to the least loaded machine. The sequence of next jobs will follow the following principle: the size of the next job will satisfy the requirement that as it is assigned to the least loaded machine (always machine m), the least loaded machine will be the most loaded machine (reposition to machine 1), and the ratio between the load of every two consecutive machines is remained. The following illustrated that principle:





For such sequence of jobs and principle to be obeyed, we conjecture that the optimal schedule (OPT) can be obtained by sorting the job sequence in reversed order and assigning them using List Scheduling Algorithm. Our simulation shows that this is true:

Figure 1: Algorithm's schedule

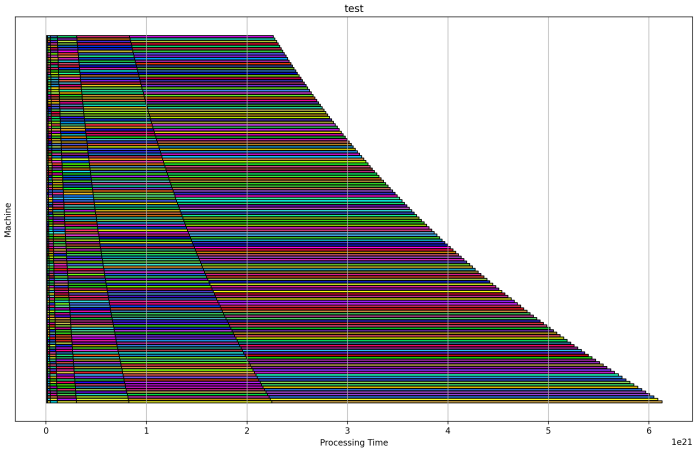
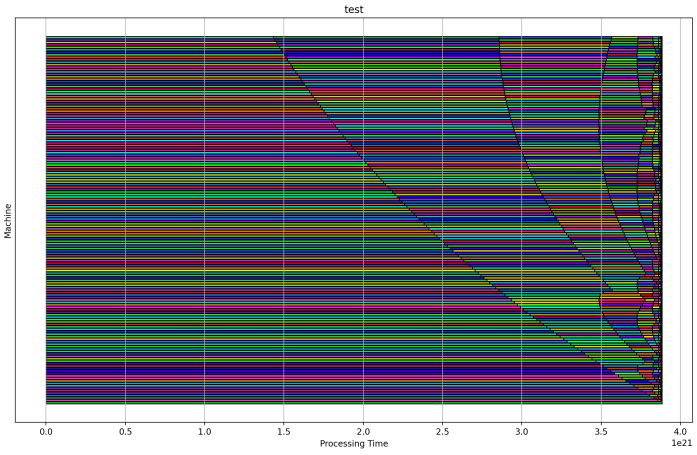


Figure 2: Optimal schedule



Our simulation shows that the OPT will always be the largest job, which is also the very last job being assigned to a machine. Also, if the OPT is true, then as we increase the number of machines and number of phases of adding job, the lower bound converges to $\frac{e}{e-1}$

3.3 Proof for the lower bound:

Lemma 4. *Let S be a sequence of n job* $(1,2,\ldots,n)$ *such that for every job* $j > m$, $p_j \frac{\sum_{i=1}^{j-1} p_i}{m-1}$ *and* $p_m = \varepsilon p_n$ *where* ε *is a very small number. Then, by greedily assigning jobs using Graham List Scheduling Algorithm in the job order of* $(n, n-1, \ldots, 2, 1)$ *produces a makespan of*

$$\frac{\sum_{i=m+1}^{n} p_i}{m} + \max_{0 \le i \le m} p_i$$

Proof. Notice that by having a sequence following such constraint, it is true that when each job in the sequence $(n, n-1, \ldots, m+1)$ is assigned, the makespan remains under the average volume of the sequence. We can prove this by using proof of contradiction. FSOC, assuming that there exist a job k such that when it is assigned to a machine, the makespan exceeds the average volume.
Because of the greedy assignment of jobs, the load of the machine k' that job k is assigned to must be the least before it is assigned k. Therefore, the load of every machine other than k' must greater than the load of k' before it is assigned k. However, since we know that for every job $j > m$, $p_j \frac{\sum_{i=1}^{j-1} p_i}{m-1}$, so for every other job $k-1, k-2, \ldots, m+1$, even if we can assign them evenly among $m-1$ machines, the overall makespan must always be greater than the average volume. Thus, this gives us a contradiction.
Because of this, for the sequence $n, n-1, \ldots, m+1$ being assigned, the makespan always at most the average volume, or $\frac{\sum_{i=m+1}^{n} p_i}{m}$.
Regarding the remaining m jobs, their size is very small with respect to the average volume, so adding them to the previous schedule only make the makespan to be at most $\frac{\sum_{i=m+1}^{n} p_i}{m} + \max_{0 \leq i \leq m} p_i$. □ □

References

[1] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[2] S. Guo and L. Kang. Online scheduling of malleable parallel jobs on two identical machines. *European Journal of Operational Research*, 206(3):555–561, 2010.

[3] J. T. Havill. Online malleable job scheduling for m $\leq$ 3. *Information Processing Letters*, 111(1):31—-35, 2010.

[4] 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.

[5] Nathaniel Kell and Jessen Havill. Improved upper bounds for online malleable job scheduling. *Journal of Scheduling*, 18(4):393–410, Aug 2015.

[6] John F Rudin III and Ramaswamy Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32(3):717–735, 2003.