# Weighted completion time minimization for capacitated parallel machines

Ilan Reuven Cohen[1], Izack Cohen[2], and Iyar Zaks[3]

[1] Faculty of Engineering, Bar-Ilan University, Israel. `ilan-reuven.cohen@biu.ac.il`
[2] Faculty of Engineering, Bar-Ilan University, Israel. `izack.cohen@biu.ac.il`
[3] Faculty of Industrial Engineering and Management, The Technion - IIT, Israel. `iyarzaks@gmail.com`

**Abstract.** We consider the weighted completion time minimization problem for capacitated parallel machines, which is a fundamental problem in modern cloud computing environments. We study settings in which the processed jobs may have varying duration, resource requirements and importance (weight). Each server (machine) can process multiple concurrent jobs up to its capacity. Due to the problem's $\mathcal{NP}$-hardness, we study heuristic approaches with provable approximation guarantees. We first analyze an algorithm that prioritizes the jobs with the smallest volume-by-weight ratio. We bound its approximation ratio with a decreasing function of the ratio between the highest resource demand of any job to the server's capacity. Then, we use the algorithm for scheduling jobs with resource demands equal to or smaller than 0.5 of the server's capacity in conjunction with the classic weighted shortest processing time algorithm for jobs with resource demands higher than 0.5. We thus create a hybrid, constant approximation algorithm for two or more machines. We also develop a constant approximation algorithm for the case with a single machine. This research is the first, to the best of our knowledge, to propose a polynomial-time algorithm with a constant approximation ratio for minimizing the weighted sum of job completion times for capacitated parallel machines.

## 1  Introduction

In this work, we study capacitated machine scheduling problems. These problems were initially encountered in production settings where jobs are processed in batches (e.g., scheduling jobs for heat treatment ovens and wafer fabrication processes [5]). Recently, interest in these problems has increased because of the search for solutions that can be used in modeling modern cloud computing environments [7]. Differently than most scheduling models in which a resource serves a single job at any given time [2,4,9,23], in modern cloud computing environments, multiple jobs can run concurrently on the same server subject to its capacity constraints (e.g., memory, cores, bandwidth). This fact can be seen in the following three examples of well-known cloud computing platforms in which resources are simultaneously shared by multiple jobs and clients: Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP). These platforms leverage virtualization technologies, such as VMware products and Xen, to allow each physical machine to be shared by multiple jobs. Virtualization also helps in reducing the costs of maintenance, operation and provisioning [20].

Managers of cloud computing environments who wish to increase the utilization of their data centers typically resort to improving the scheduling algorithms that allocate jobs to machines. In this context, we focus on minimizing the weighted sum of job completion times, which is one of most common objective functions [7]. The weights imply that some jobs may be more important than others; thus, the scheduler has to take into consideration that the delay of one job can incur a higher "cost" than the delay of another.

The non-capacitated counterparts of the considered problem have been widely researched. One of the most known results is that the Shortest Processing Time (SPT) priority rule minimizes the (non-weighted) sum of job completion times [23]. The SPT was extended to the Weighted Shortest Processing Time (WSPT), which was used in the weighted version of the completion time minimization problem. Since the latter problem is $\mathcal{NP}$-complete for more than two machines [8], various solution approaches focus on developing polynomial-time heuristic approximation algorithms that bound the worst case performance with respect to an optimal solution. For concreteness, a desired algorithm $A$ for a minimization problem $P$ would have a $\rho$ approximation ratio such that for any instance $I$ of $P$, $A(I) \leq \rho \cdot OPT(I)$, where $OPT(I)$ is the value of an optimal solution for $I$. At [6] they proved that scheduling according to the WSPT priority rule provides a constant approximation algorithm. [16] improved the WSPT approximation ratio $(\rho)$ to $(1 + \sqrt{2})/2$ and proved that it is tight.

In the capacitated setting, [12] was the first to present a constant approximation algorithm for minimizing the *non-weighted* sum of completion times using the Smallest Volume First (SVF) priority rule combined with the SPT priority rule. Our work extends theirs into the more general, weighted case, by combining Weighted Smallest Volume First (WSVF) and WSPT with the objective of developing the first algorithm with a constant approximation

ratio for the weighted completion time problem in the capacitated setting. Our analysis, moreover, improves their approximation ratio for the non-weighted case.

## 1.1   Contributions and Techniques

We develop a constant approximation ratio algorithm for minimizing the weighted sum of job completion times on capacitated machines. Here are the main contributions:

1. A polynomial-time scheduling algorithm with a $\left(1 + \frac{1}{1-\alpha}\right)$-approximation ratio, if the ratio between jobs' demands and the servers' capacities is at most $\alpha$.
2. A polynomial-time scheduling algorithm with a $4 + o(1/M)$ approximation ratio for $M \geq 2$ machines.
3. A polynomial-time scheduling algorithm with a $12 + \epsilon$ approximation ratio for a single machine ($M{=}1$).

To the best of our knowledge, our result is the first constant approximation algorithm for the weighted completion time minimization problem for the capacitated parallel machine problem. In addition, we improve the approximation guarantees for the non-weighted version. In [12], the authors proved a $\left(\frac{3\alpha}{1-\alpha} + 3\right)$ approximation ratio, for the case when the ratio of jobs' demands and servers' capacities is at most $\alpha$, and a $(5 + o(1/M))$ approximation ratio for $M > 1$. The new way we analyse the algorithm for $M > 1$ and the use of lemmas from [16] enable us to prove the approximation ratio for the weighted case and improve the ratio presented at [12] for the unweighted case. Our algorithm for $M > 1$ partitions the jobs into high- and low-resource demand classes where jobs within the former class require 50% or more of a machine's capacity and jobs in the latter class require less than 50%. Accordingly, the algorithm partitions the machines into two groups for processing the two job classes. The high-demand class is scheduled via the WSPT and the low-demand class via a WSVF priority rule. In the WSVF method, jobs are ordered in a non-decreasing order of the ratio between their processing time multiplied by the demand to the weight. Then, the jobs are assigned to a machine according to their priority, with the algorithm assigning the next unscheduled job to the earliest possible time $t$. Our proof involves analyzing the WSVF performance for instances where job resource demands are smaller than a constant $\alpha$.

In the analysis of the WSVF, we bound the start time of each job, provided that prior to the start time, all the machines processed at least $1 - \alpha$ demand of higher priority jobs. Then, we bound the optimal cost, by the optimal cost of a non-capacitated converted instance. Finally, we develop an improved bound by using the characterization of [6] for the non-capacitated setting. For the single machine case, we can not use the partition base approach so we extend the algorithm from [12] to the weighted case (the full details for this case are presented in the appendix).

## 1.2   Prior Work

There is a vast amount of research about machine scheduling problems owing to their theoretical and practical importance (interested readers are referred to the following reviews [18, 21]. For the sake of brevity, we focus on recent capacitated machine scheduling studies.

Researchers explored several objective functions. One very popular objective is to minimize the processing makespan—that is, the completion time of the last job (e.g., [3], [22], [13] and [14]). Others suggested that in settings with a release time and deadline for each job, a reasonable objective is to maximize the total weight of jobs completed before their deadline (see [1] and [10]). Another line of research models the capacitated machine scheduling problem as an online problem in which jobs arrive over time [17]. In this line of research, typical objective functions are to minimize the response time (i.e., the time elapsed from the job's arrival until it is scheduled) and to maximize the throughput [11].

Three works that considered the weighted sum of flow-times or completion times are [7], [12] and [19]. At [7] the researchers considered an online problem of weighted flow time minimization, assuming that jobs can be pre-empted with no penalty and delay. It is important to note, though, that preemption may incur significant switching costs (e.g., setup costs) and memory loss; preemption may be also forbidden due to system restrictions or client commitments.

[19], who also studied an online capacitated machine scheduling problem, assumed that a job can run at a slower rate when receiving a fraction of its demand or that a job can be processed in parallel on different machines. They then used Online Convex Optimization (OCO) to solve the scheduling optimization problem. These assumptions may hold in specialized computing environments but in standard environments it may be costly or technically infeasible to split a job between machines or to process it at a slower rate using a portion of the required resources.

[12] proposed a constant approximation algorithm for minimizing the sum of completion times. They, however, did not consider the more general weighted version of the problem. We close this gap by developing an approximation

algorithm that solves the weighted version of the problem and improves the approximation ratio for the non-weighted version that was presented in [12].

## 2   Formal Problem Definition

We consider $N$ jobs that need to be processed by $M$ identical machines. Each job $j$ has a processing time $p_j$, demand $d_j$ and weight $w_j$, which are known in advance. We assume, without loss of generality (hereafter, w.l.o.g.), that $p_j \geq 1$, and $d_j \in (0,1]$ is a fraction of the required demand with respect to a machine's capacity. We denote $v_j = p_j \cdot d_j$ as job $j$'s volume. We focus on a non-preemptive schedule, meaning that a started job is processed without interruption until its completion. The scheduler assigns each job to a machine and determines its start time, $s_j$; accordingly, the completion time of the job is $c_j = s_j + p_j$. We mark $T$ as the upper bound on the maximal completion time of a job (details about how to set it appeared later in the article), For convenience We mark $\mathcal{N} = \{1, 2, ..., N\}, \mathcal{M} = \{1, 2, ..., M\}, \mathcal{T} = \{0, 1, ..., T\}$.

Let $G^i(t)$ be the set of jobs processed by machine $i$ at time $t \in \mathcal{T}$. $j \in G_i(t)$ if job $j$ is assigned to machine $i$ and $t \in [s_j, c_j)$. A feasible schedule must ensure that the total demand of the jobs assigned to a machine does not exceed its capacity, at any given time. Mathematically,

$$\sum_{j \in G^i(t)} d_j \leq 1 \quad \forall i \in \mathcal{M}, t \in \mathcal{T}.$$

Our goal is to find a feasible solution that minimizes the weighted sum of completion times:

$$\min \sum_{j \in \mathcal{N}} w_j \cdot c_j.$$

As mentioned, the problem is $\mathcal{NP}$-complete. Accordingly, we are looking for a polynomial-time scheduling algorithm with a guarantee on the maximal ratio between the objective function value achieved by the algorithm and the optimal solution value. We want the developed approximation algorithm to provide a constant ratio. As discussed next, we base our algorithm on the WSVF priority rule.

## 3   WSVF Algorithm and Analysis

The WSVF algorithm orders the jobs in a non-decreasing order according to their volume over weight values, i.e., $(p_j \cdot d_j)/w_j$. The algorithm schedules the highest priority unassigned job (the one with the smallest value) at the earliest time $t$ on a machine that is available to process the job until it is completed; see Figure 1 for an example.

---
**Algorithm 1:** WSVF
***
    Set $\mathcal{J}$ to be list of jobs, sorted in a non-decreasing order of their $(p_j \cdot d_j)/w_j$ values
    **for** job $j \in \mathcal{J}$ **do**
        Schedule job $j$ on machine $i$ that provides the earliest start time $t$, s.t.

$$d_j + \sum_{h \in G_i(t')} d_h \leq 1, \text{ for all } t' \in [t, t + p_j)$$

    **end for**

---

By definition, the assignment of Algorithm 1 is feasible. Next, we show that its approximation ratio depends on $\alpha = \max_{j \in \mathcal{N}} d_j$, the maximum resource demand of any job.
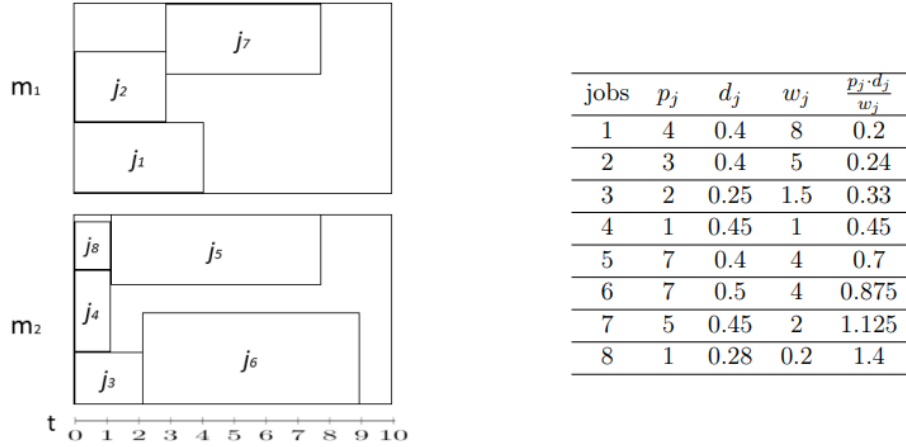
| jobs | $p_j$ | $d_j$ | $w_j$ | $\frac{p_j \cdot d_j}{w_j}$ |
|------|-------|-------|-------|------------------------------|
| 1 | 4 | 0.4 | 8 | 0.2 |
| 2 | 3 | 0.4 | 5 | 0.24 |
| 3 | 2 | 0.25 | 1.5 | 0.33 |
| 4 | 1 | 0.45 | 1 | 0.45 |
| 5 | 7 | 0.4 | 4 | 0.7 |
| 6 | 7 | 0.5 | 4 | 0.875 |
| 7 | 5 | 0.45 | 2 | 1.125 |
| 8 | 1 | 0.28 | 0.2 | 1.4 |

**Fig. 1.** An illustration of the WSVF algorithm's run. In this illustration we have $M = 2$ machines and $N = 8$ jobs. Each job is defined by its parameters $(p_j, d_j, w_j)$ as described above. The jobs are sorted by a non-decreasing order of $p_j \cdot d_j / w_j$, for example, $p_1 \cdot d_1 / w_1 = 0.2 < 0.24 = p_2 \cdot d_2 / w_2$. Then running over the jobs according to the priority order, each is scheduled on the machine with the earliest available $d_j$ capacity for $p_j$ time steps. Here, jobs $j_1, j_2, j_3$ and $j_4$ are scheduled at time 0 according to the available machine capacities. Job $j_5$ can be scheduled only after the completion of $j_4$ so $s_5 = c_4 = 1$, and jobs $j_6$ and $j_7$ are waiting for the completion of jobs $j_3$ and $j_2$, respectively. Job $j_8$, although it has a low priority, is scheduled at time 0 because machine $m_2$ has a resource 'window' that is large enough to process job $j_8$. The objective function value in the example is $\sum_{j=1}^{8} w_j \cdot c_j = 8 \cdot 4 + 5 \cdot 3 + \cdots + 2 \cdot 8 + 0.2 \cdot 1 = 135.2$.

**Theorem 1.** *If any job requires at most $\alpha < 1$, WSVF is a $\left(\frac{1}{1-\alpha} + 1\right)$-approximation algorithm for the weighted completion time minimization problem.*

To prove Theorem 1, we need to establish bounds on a problem instance $\hat{I}$, which is a compressed instance of the original problem instance $I$, where $I$ denotes the set of jobs $\mathcal{J}$ ordered by the WSVF priority rule. We "compress" the original jobs such that their demands become 1 (as in the non-capacitated setting). For this, we take the ordered set of jobs $I$, and for each job in the set, we fix the duration to be $\hat{p}_j = p_j \cdot d_j$, which is the so-called volume of job $j$ in the original instance, and fix the demand as $\hat{d}_j = 1$. We note that the compressed instance preserves the ordering of the jobs and the list of jobs is ordered according to the WSPT priority rule.

We bound the cost of WSVF on $I$ by proving that the optimal solution value of the corresponding compressed instance $\hat{I}$ is smaller than or equal to the optimal solution value of $I$. For ease of notation, we denote by $C_M(I)$ the objective function value (cost) achieved by algorithm WSVF for instance $I$ using $M$ machines and by $C_M^*(I)$ the optimal cost of scheduling $I$ on $M$ machines.

We note the following two observations: 1) $C_N(I) = \sum_{j \in \mathcal{N}} w_j p_j$ since when using $N$ machines and $N = |I|$, it is always optimal to assign each job to a machine, and 2) for a single machine, $C_1(\hat{I}) = C_1^*(\hat{I}) = \sum_j (w_j \cdot \sum_{h=1}^{j} \hat{p}_h)$ by Smith's rule [24].

We begin by establishing the relations between $C_M^*(I), C_M^*(\hat{I}), C_1^*(\hat{I})$ and $C_M(I)$. First, since $c_j \geq p_j$ always holds:

**Observation 2** $C_N(I) \leq C_M^*(I)$.

Next, we bound the optimal cost on $M$ machines by the optimal cost on the compressed instance on $M$ machines.

**Claim 1** $C_M^*(\hat{I}) \leq C_M^*(I)$.

*Proof.* Consider, w.l.o.g., one of the $M$ machines and let $j_1, j_2, ...., j_k$ be the jobs assigned to this machine ordered by their completion times in the optimal schedule (which has a cost $C_M^*(I)$). We prove that a scheduler that processes the compressed jobs on the same machine in this order has a smaller than or equal to cost compared to the uncompressed optimal instance. Let $c_j^*$ be the completion time of job $j$ in the optimal original instance, and $\hat{c}_j$ be its completion time in the compressed instance schedule. We will show that $c_j^* \geq \hat{c}_j$. First, observe that $\hat{c}_j = \sum_{h=1}^{j} \hat{p}_h$, since in the compressed schedule, the machine can process a single job at each time. Next, by

definition, at time $c_j^*$, all the first $j$ jobs have been completed on this machine. By the volume preservation rule, it took at least $\sum_{h=1}^{j} p_h \cdot d_h$ time units to complete the jobs since at any time step, the machine can process a maximal volume of 1; therefore, $c_j^* \geq \sum_{h=1}^{j} p_h \cdot d_h = \sum_{h=1}^{j} \hat{p}_h = \hat{c}_j$. Hence, when we sum over all machines and jobs,

$$C_M^*(\hat{I}) \leq \sum_{j \in \mathcal{N}} w_j \cdot \hat{c}_j \leq \sum_{j \in \mathcal{N}} w_j \cdot c_j^* = C_M^*(I).$$

The Left-Hand-Side (LHS) inequality holds since the optimal cost $C_M^*$ is equal to or smaller than the cost of any schedule and the Right-Hand-Side (RHS) inequality follows from extending the volume preservation argument to $M$ machines.

Next, we use Theorem 1 from [6] to construct a lower bound for $C_M^*(\hat{I})$ using a single machine optimal cost $C_1^*(\hat{I})$.

**Claim 2** $C_M^*(\hat{I}) \geq \frac{1}{M} C_1^*(\hat{I})$.

*Proof.* [6] proved a tight bound on the optimal cost of the non-capacitated case with $M$ machines with respect to the optimal cost of an instance with a single machine. Specifically, they proved

$$C_M^*(\hat{I}) - \frac{1}{2} C_N(\hat{I}) \geq \frac{1}{M}(C_1^*(\hat{I}) - \frac{1}{2} C_N(\hat{I})).$$

Therefore, we have

$$C_M^*(\hat{I}) \geq \frac{1}{M} C_1^*(\hat{I}) + \frac{1}{2} C_N(\hat{I})(1 - \frac{1}{M}) \geq \frac{1}{M} C_1^*(\hat{I})$$

since $\frac{1}{2} C_N(\hat{I})(1 - \frac{1}{M}) \geq 0$.

We now compare $C_M(I)$ and $C_1(\hat{I})$ by bounding the start time $s_j$ for each job $j \in \mathcal{N}$. We prove that $s_j$ can be bounded by a factor that depends on $\alpha$ and $M$ times the total volume of jobs preceding $j$ in $I$ (which is ordered by WSVF). In other words, $\sum_{h<j} v_h$, which is the start time of job $j$ in $C_1(\hat{I})$.

**Lemma 3.** $\forall j \in \mathcal{N} : s_j \leq \frac{1}{(1-\alpha)M} \Sigma_{h=1}^{j-1} v_h$

*Proof.* To prove Lemma 3, we prove that prior to $s_j$, each machine processes a total demand of at least $1 - \alpha$ on jobs with a higher priority than $j$. This property also appeared in [12] and holds for any priority-based algorithm. For a machine $i \in \mathcal{M}$ at time $t \in \mathcal{T}$ and job $j \in \mathcal{N}$, we define $D_j^i(t) = \sum_{h \in G^i(t), h \leq j} d_h$, which is the total demand of jobs among the first $j$ jobs that are processed on machine $i$ at time $t$. By proving the following property, we prove that for all $t < s_j$, we have $D_{j-1}^i(t) \geq 1 - \alpha$.

**Claim 4** *For all $i \in \mathcal{M}, j \in \mathcal{N}$, $min\{D_j^i(t), 1 - \alpha\}$ is non-increasing with $t$.*

*Proof.* Choose a machine $i$ (we omit the superscript $i$ when it is clear from the context), and consider w.l.o.g. only jobs assigned to this machine (with the same priority order as in $I$), other jobs not affected the total demand $D$. We prove the claim using an induction on $j$. First, consider the case of $j = 1$,

$$min\{D_1(t), 1 - \alpha\} = \begin{cases} min\{d_1, 1 - \alpha\} & \text{if } t \leq p_1, \\ 0 & \text{if } t > p_1. \end{cases}$$

It is a non-increasing function with $t$ so the claim holds for $j = 1$.
Now we use the induction assumption that the claim is true for $j - 1$ and prove the claim for $j > 1$. First, we argue that according to the WSVF, $D_j(t) = D_{j-1}(t) > 1 - \alpha$, for all $0 \leq t < s_j$. Otherwise, if $0 \leq t < s_j$ exists such that $D_{j-1}(t) \leq 1 - \alpha$ (that is, $D_{j-1}(t) + \alpha \leq 1$), then for $t \leq t' < s_j$, we have $D_{j-1}(t') \leq 1 - \alpha$ by our induction assumption, and since $d_j \leq \alpha$ by our assumption on the input, the WSVF would assign job $j$ before time $s_j$.
Now, we can look at the expression $min\{D_j(t), 1 - \alpha\}$ value over time $t$:

$$min\{D_j(t), 1 - \alpha\} = \begin{cases} 1 - \alpha & \text{if } t < s_j, \\ min\{D_{j-1}(t) + d_j, 1 - \alpha\} & \text{if } s_j \leq t \leq s_j + p_j, \\ min\{D_{j-1}(t), 1 - \alpha\} & \text{if } t > s_j + p_j. \end{cases}$$

Using the above and the induction assumption, we can conclude that $min\{D_j(t), 1 - \alpha\}$ is non-increasing with $t$.

**Corollary 1.** *For all $i \in \mathcal{M}, j \in \mathcal{N}$ and $0 \leq t < s_j$, we have $D_{j-1}^i(t) > 1 - \alpha$.*

*Proof.* We follow the proof of Claim 4; if at any $t < s_j$, $D_{j-1}(t) \leq 1 - \alpha$, then by the invariant of Claim 4, the WSVF would process job $j$ earlier than $s_j$ by its definition.

Next, we use Corollary 1 to set an upper bound on the starting time of every job $j$ and to conclude the proof of Lemma 3. By corollary 1, we have that each machine used at least $(1 - \alpha)$ of its capacity until time $s_j$, to process the first $j - 1$ jobs. Thus, the following inequality holds for $M$ machines

$$\Sigma_{h=1}^{j-1} v_h \geq M(1 - \alpha)s_j,$$

where the LHS is the sum of the first $j - 1$ job volumes that were processed on $M$ machines and the RHS follows from the lower bound on the used volume extended to $M$ machines. Reorganizing the above formula leads to an upper bound on the starting time of job $j$:

$$s_j \leq \frac{1}{(1 - \alpha)M} \Sigma_{h=1}^{j-1} v_h.$$

By using the above results, we can prove the main lemma for bounding $C_M(I)$:

**Lemma 5.** *Given any instance $I$, such that for all $j \in \mathcal{N}$, $d_j \leq \alpha$ for $0 < \alpha < 1$ we have:*

$$C_M(I) \leq C_N(I) + \frac{C_1(\hat{I})}{(1 - \alpha)M}$$

*Proof.* For every job $j \in \mathcal{N}$

$$c_j = p_j + s_j \leq p_j + \frac{1}{(1 - \alpha)M} \cdot \sum_{h=1}^{j-1} v_h = p_j + \frac{1}{(1 - \alpha)M} \cdot \sum_{h=1}^{j-1} \hat{p}_h, \tag{1}$$

where the LHS equality exists by definition, the inequality follows from Lemma 3, and the RHS equality follows from the definition $v_h = d_h \cdot p_h = \hat{p}_h$.
Summing over all jobs to find the total cost, we have:

$$
\begin{aligned}
C_M(I) &= \sum_{j=1}^{N} w_j \cdot c_j \\
&\leq \sum_{j=1}^{N} w_j \cdot p_j + \sum_{j=1}^{N} w_j \cdot \left( \frac{1}{(1 - \alpha)M} \cdot \sum_{h=1}^{j-1} \hat{p}_h \right) \\
&= \sum_{j=1}^{N} w_j \cdot p_j + \frac{1}{(1 - \alpha)M} \cdot \sum_{j=1}^{N} (w_j \cdot \sum_{h=1}^{j-1} \hat{p}_h) \\
&\leq C_N(I) + \frac{C_1(\hat{I})}{(1 - \alpha)M},
\end{aligned}
$$

where the first inequality follows from Equation 1 and the second inequality holds since $C_N(I) = \sum\limits_{j=1}^{n} w_j \cdot p_j$, and $C_1(\hat{I}) = \sum\limits_{j=1}^{n} (w_j \cdot \sum\limits_{h=1}^{j} \hat{p}_h) \geq \sum\limits_{j=1}^{n} (w_j \cdot \sum\limits_{h=1}^{j-1} \hat{p}_h)$.

Finally, we prove Theorem 1, which follows immediately from Lemma 5 and Claims 1 and 2.

*Proof (Proof of Theorem 1).*

$$C_M(I) \leq C_N(I) + \frac{C_1(\hat{I})}{(1 - \alpha)M} \leq C_N(I) + \frac{C_M^*(\hat{I})}{1 - \alpha} \leq C_M^*(I) \left( 1 + \frac{1}{1 - \alpha} \right).$$

From the above we can deduce that for high load systems ($C_N(I) << C_M^*(I)$) with low demands jobs ($\alpha << 1$) WSVF getting closer to the optimal solution.

## 4    Constant Approximation Algorithm for all Instances

In this section, we extend the previous results to remove the dependency on $\alpha$, the maximum demand of any job, which leads to a constant approximation algorithm. We rely on the observations that, given a set of jobs with demands that are larger than $1/2$, only a single job can run on a machine at any time, and that the WSVF has a constant approximation ratio on jobs with demands equal to or smaller than $1/2$. Thus, by splitting the jobs based on their resource demands into two sets of machines, we achieve a constant approximation algorithm.

To this end, we introduce the Hybrid-WSVF:

---
**Algorithm 2:** HYBRID-WSVF
---
Split the jobs in $I$ into two sets $I^l = \{j : d_j \leq \frac{1}{2}\}$ and $I^h = \{j : d_j > \frac{1}{2}\}$.

Schedule $I^l$ on $M_1 = \lceil \frac{2(M-2)}{3} \rceil + 1$ machines using WSVF.

Schedule $I^h$ on $M_2 = M - M_1$ using WSPT.

---

**Theorem 3.** *HYBRID-WSVF is a $4 + o(\frac{1}{M})$-approximation for the weighted completion time minimization problem where $M$ is the number of machines.*

*Proof.* First, we schedule the low demand jobs (i.e., $d_j \leq 0.5$) on $M_1$ machines. Thus, from Lemma 5, we have

$$C_{M_1}(I^l) \leq C_N(I^l) + \frac{1}{(1-\alpha)M_1} \cdot C_1(\hat{I}^l).$$

For the low demand jobs, $\alpha$ values are within the interval $(0, 0.5]$ with a specific value per $I^l$. Thus,

$$C_N(I^l) + \frac{1}{(1-\alpha)M_1} \cdot C_1(\hat{I}^l) \leq C_N(I^l) + \frac{2}{M_1} \cdot C_1(\hat{I}^l)$$

$$= C_N(I^l) + \frac{2M}{M_1} \cdot \frac{C_1(\hat{I}^l)}{M} \leq C_M^*(I^l) + 2 \cdot \frac{M}{M_1} C_M^*(I^l),$$

where the last RHS inequality follows from Observation 2, Claim 1 and Claim 2.

Next, we deal with the high demand jobs ($d_j > 0.5$), which are scheduled on $M_2$ machines. Since any two high demand jobs cannot run simultaneously on the same machine, the scheduling problem is equivalent to the weighted non-capacitated setting. Therefore, we can use a result from [6] who proved the following bound for the WSPT with $M$ machines:

$$C_M(\hat{I}) \leq C_N(\hat{I}) + \frac{1}{M} C_1(\hat{I}).$$

Following their result, we state that:

$$C_{M_2}(I^h) \leq C_N(I^h) + \frac{1}{M_2} C_1(I^h) = C_N(I^h) + \frac{M}{M_2} \frac{C_1(I^h)}{M} \leq C_M^*(I^h) + \frac{M}{M_2} C_M^*(I^h),$$

where the RHS inequality follows from Observation 2 and Claim 2.

By setting $M_1 = \lceil \frac{2(M-2)}{3} \rceil + 1$ and $M_2 = M - M_1 = \lfloor \frac{M-2}{3} \rfloor + 1$, and by observing $C_M^*(I) \geq C_M^*(I^h) + C_M^*(I^l)$, we conclude the proof for Theorem 3.

## 5    Conclusions and Future Work

Algorithms for scheduling jobs on capacitated machines can significantly affect the performance of cloud computing environments. While such environments handle jobs of varying importance (e.g., cost), there were no constant approximation algorithms known, to the best of our knowledge, for the related problem of minimizing the weighted sum of job completion times. This paper closes this gap. The suggested algorithm also improves the best-known approximation ratio for the non-weighted problem (for $M \geq 2$).

The results presented in this paper may be enhanced by going in several research directions, of which we mention three. The first significant theoretical extension may consider capacitated problems with multiple capacitated resources. In real-world cloud computing environments, CPU, memory, storage and bandwidth may be scarce resources. To realize such an extension, one would need to consider a multidimensional demand for each job. A second extension would be to use our methods to find performance guarantees in stochastic environments in which, for example, processing duration can be characterized using probabilistic knowledge. The third research direction would be to develop a model that accommodates jobs with release dates and deadlines. Such research may also be considered to be a natural extension of the current model.

## Appendix: The single machine case

In this section, we introduce a different algorithm for the special case of $M = 1$, and conclude that for the weighted completion time minimization for capacitated machine, there exists a constant competitive algorithm for any number of machines. Note that Algorithm 2 is well defined for $M \geq 2$ machines; for $M = 1$ machines, the partition approach cannot be applied. Instead, we extend the ideas of the non-weighted case for $M = 1$ in [12] to the general weighted case.

The algorithm uses two main tools, a knapsack algorithm that determines a subset of highest weight jobs with a total volume restriction, and a 2D-strip packing algorithm for scheduling this set, where each job $j$ is represented by a rectangle $r_j$ defined with width $p_j$ and height $d_j$.

The algorithm will work in iterations until it assigns all jobs, where in iteration $\ell$ the algorithm will execute the following steps:

1. Solve a knapsack packing problem: Compute $J_\ell$, a maximum weighted set of jobs with a total volume less than $2^\ell$ using a $1 + \epsilon$ resource augmentation knapsack algorithm.
2. Pack the jobs in $J_\ell$ into 3 strips of width $(1 + \epsilon)2^\ell$ and height 1
3. Concatenate the strips from the earlier step and schedule the job on the machine.

Jobs can be schedule more than once when using the above algorithm, of course it is unneeded and practically the job will be scheduled only according to the first strip it appear in. Note that after $\ell^{\max} = \log(\sum_{j \in \mathcal{N}} v_j)$ iterations, all the jobs are packed, and the algorithm can discard jobs that were scheduled in previous iterations. We will show that the first two steps of finding the set of jobs and packing it into strips can be done in polynomial time. Finding a maximum weighted set of jobs ($\arg \max_J \{\sum_{j \in J} w_j : \sum_{j \in J} v_j \leq B\}$) is equivalent to the knapsack problem: Given a set of items, each with a size and a value, determine which item to include in a collection so that the total size is less than or equal to a given limit and the total value is as large as possible. We apply the resource augmentation solution of [25] for this problem, which proves that given a bound on the total size $L$, there exists a polynomial time in which a set of total size $(1 + \epsilon) \cdot L$ with a total profit of at least the optimal profit of a set with total size $L$ can be computed. Second, we utilize the algorithm from [15], which packs rectangles (without rotations) with total volume of $L$ and maximal height of 1 into 3 strips of width $L$ and height 1.

---
**Algorithm 3:** PackAndSchedule($I$)

    **for** $\ell = 0, 1, 2, \ldots, \ell^{\max}$ **do**

        Compute $J_\ell \subseteq I$, a maximal weighted set with total volume of at most $2^\ell \cdot (1 + \epsilon)$.

        Pack $J_\ell$ into strip $S$ of height 1 and width $3 \cdot (1 + \epsilon) \cdot 2^\ell$.

        Schedule $S$ in the interval $[3(1 + \epsilon) \sum\limits_{h=0}^{\ell-1} 2^h, 3(1 + \epsilon) \sum\limits_{h=0}^{\ell} 2^h)]$

    **end for**

---

**Theorem 4.** *PackAndSchedule is a $12(1+\epsilon)$ approximation polynomial-time algorithm for total weighted completion time minimization in the single machine case.*

*Proof.* The algorithm provides a feasible scheduling allocation since each of the 2D packing solution strips having a maximal height (demand) of 1 is assigned to a disjoint time interval.

Let $W_\ell = \sum_{j \in J_\ell} w_j$ be the sum of weights that are processed in iteration $\ell$. Note that jobs in $J_\ell$ are scheduled until time $3(1 + \epsilon) \cdot (2^{\ell+1} - 1)$. In addition, when jobs in $J_\ell$ are processed, the total weight of jobs that are not yet completed is at most $W - W_{\ell-1}$. Therefore, we can bound the cost of the algorithm 3 by summing over the iterations and multiplying the unprocessed weight by the completion time:

$$C_1(I) \leq 3(1 + \epsilon) \sum_{\ell \geq 0} (2^{\ell+1} - 1)(W - W_{\ell-1})$$

The optimal scheduler can only complete jobs with a total weight of at most $W_\ell$ up to time $2^\ell$, because by volume preservation, it is not possible to pack more than $2^\ell$ volume until this point, and by the correctness of the knapsack algorithm, this is the maximum profit (weight) that can be packed by $W_\ell$ if the total volume is at most $2^\ell$; therefore, there are jobs with a total weight of at least $W - W_\ell$ that are unprocessed at time $2^\ell$. Hence, we can give a lower bound for the cost of the optimal scheduler:

$$C_1^*(I) \geq W + \sum_{\ell \geq 0} 2^\ell (W - W_\ell)$$

By combining the two above inequalities, we have that PackAndSchedule is a $12(1 + \epsilon)$ approximation algorithm as required.

# References

1. Albagli-Kim, S., Shachnai, H., Tamir, T.: Scheduling jobs with dwindling resource requirements in clouds. In: IEEE INFOCOM 2014-IEEE Conference on Computer Communications. pp. 601–609. IEEE (2014)
2. Balouka, N., Cohen, I.: A robust optimization approach for the multi-mode resource-constrained project scheduling problem. European Journal of Operational Research (2019)
3. Bougeret, M., Dutot, P.F., Jansen, K., Robenek, C., Trystram, D.: Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms. Discrete Mathematics, Algorithms and Applications **3**(04), 553–586 (2011)
4. Cohen, I., Postek, K., Shtern, S.: An adaptive robust optimization model for parallel machine scheduling. arXiv preprint arXiv:2102.08677 (2021)
5. Damodaran, P., Ghrayeb, O., Guttikonda, M.C.: Grasp to minimize makespan for a capacitated batch-processing machine. The International Journal of Advanced Manufacturing Technology **68**(1-4), 407–414 (2013)
6. Eastman, W.L., Even, S., Isaacs, I.M.: Bounds for the optimal scheduling of n jobs on m processors. Management science **11**(2), 268–279 (1964)
7. Fox, K., Korupolu, M.: Weighted flowtime on capacitated machines. In: Proceedings of the twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 129–143. SIAM (2013)
8. Garey, M.R., Johnson, D.S.: Computers and intractability. A Guide to the (1979)
9. Graves, S.C.: A review of production scheduling. Operations research **29**(4), 646–675 (1981)
10. Guo, L., Shen, H.: Efficient approximation algorithms for the bounded flexible scheduling problem in clouds. IEEE Transactions on Parallel and Distributed Systems **28**(12), 3511–3520 (2017)
11. Hota, A., Mohapatra, S., Mohanty, S.: Survey of different load balancing approach-based algorithms in cloud computing: a comprehensive review. Computational intelligence in data mining pp. 99–110 (2019)
12. Im, S., Naghshnejad, M., Singhal, M.: Scheduling jobs with non-uniform demands on multiple servers without interruption. In: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. pp. 1–9. IEEE (2016)
13. Jansen, K., Rau, M.: Linear time algorithms for multiple cluster scheduling and multiple strip packing. In: European Conference on Parallel Processing. pp. 103–116. Springer (2019)
14. Jansen, K., Trystram, D.: Scheduling parallel jobs on heterogeneous platforms. Electronic Notes in Discrete Mathematics **55**, 9–12 (2016)
15. Jansen, K., Zhang, G.: Maximizing the total profit of rectangles packed into a rectangle. Algorithmica **47**(3), 323–342 (2007)
16. Kawaguchi, T., Kyan, S.: Worst case bound of an lrf schedule for the mean weighted flow-time problem. SIAM Journal on Computing **15**(4), 1119–1129 (1986)
17. Kumar, M., Sharma, S.C., Goel, A., Singh, S.P.: A comprehensive survey for scheduling techniques in cloud computing. Journal of Network and Computer Applications **143**, 1–33 (2019)
18. Liu, S.: A review for submodular optimization on machine scheduling problems. In: Complexity and Approximation, pp. 252–267. Springer (2020)
19. Liu, Y., Xu, H., Lau, W.C.: Online job scheduling with resource packing on a cluster of heterogeneous servers. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. pp. 1441–1449. IEEE (2019)
20. Malhotra, L., Agarwal, D., Jaiswal, A., et al.: Virtualization in cloud computing. J. Inform. Tech. Softw. Eng **4**(2), 1–3 (2014)
21. Meiswinkel, S.: Mechanism design and machine scheduling: Literature review. On Combinatorial Optimization and Mechanism Design Problems Arising at Container Ports pp. 15–30 (2018)
22. Muter, İ.: Exact algorithms to minimize makespan on single and parallel batch processing machines. European Journal of Operational Research **285**(2), 470–483 (2020)
23. Pinedo, M.: Scheduling, vol. 5. Springer (2012)
24. Smith, W.E.: Various optimizers for single-stage production. Naval Research Logistics Quarterly **3**(1-2), 59–66 (1956)
25. Williamson, D.P., Shmoys, D.B.: The design of approximation algorithms. Cambridge university press (2011)