Annals of Operations Research (2024) 338:429–452
https://doi.org/10.1007/s10479-023-05613-x

ORIGINAL RESEARCH

A theoretical and empirical study of job scheduling in cloud computing environments: the weighted completion time minimization problem with capacitated parallel machines

Ilan Reuven Cohen[1] · Izack Cohen[1] · Iyar Zaks[2]

Received: 1 August 2022 / Accepted: 13 September 2023 / Published online: 19 October 2023

Abstract
We consider the weighted completion time minimization problem for capacitated parallel machines, which is a fundamental problem in modern cloud computing environments. In our setting, the processed jobs may be of varying duration, require different resources, and be of unequal importance (weight). Each server (machine) can process multiple concurrent jobs up to its capacity. We study heuristic approaches with provable approximation guarantees and offer an algorithm that prioritizes the jobs with the smallest volume-by-weight ratio. We bound the algorithm's approximation ratio using a decreasing function of the ratio between the highest resource demand of any job and the server's capacity. Thereafter, we create a hybrid, constant approximation algorithm for two or more machines. We also develop a constant approximation algorithm for the case of a single machine. Via a numerical study and a mixed-integer linear program of the problem, we demonstrate the performance of the suggested algorithm with respect to the optimal solutions and alternative scheduling methods. We show that the suggested scheduling method can be applied to both offline and online problems that may arise in real-world settings. 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.

Keywords Scheduling · Capacitated machines · Cloud computing · Parallel machines · Approximation algorithms

✉ Izack Cohen
izack.cohen@biu.ac.il

Ilan Reuven Cohen
ilan-reuven.cohen@biu.ac.il

Iyar Zaks
iyarzaks@gmail.com

1 Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel

2 Faculty of Industrial Engineering and Management, Technion–Israel Institute of Technology, Haifa, Israel

1 Introduction

We focus on the capacitated machine scheduling problem, an important topic in production settings where jobs are processed in batches [e.g., scheduling jobs for heat treatment ovens and wafer fabrication processes; (Damodaran et al., 2013)]. In recent years, capacitated machine scheduling techniques have been used for modeling modern cloud computing environments (Fox and Madhukar, 2013). In contrast to most scheduling models in which a server deals with a single job at any given time [e.g., (Hart et al., 2005; Pinedo, 2012; Balouka and Izack, 2021; Cohen et al., 2023)], in cloud computing environments, multiple jobs can run concurrently on the same server, subject to capacity constraints (e.g., memory). For example, resources are shared by multiple jobs and clients in three well-known cloud computing platforms: 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. Additionally, virtualization helps in reducing maintenance, operation and provisioning costs (Malhotra et al., 2014; Jain and Sakshi, 2016).

Typically, when aiming to increase the utilization of their data centers, cloud computing managers concentrate on improving the scheduling algorithms that allocate jobs to machines. We focus on minimizing the weighted sum of job completion times—one of the most common objective functions in such environments (Fox and Madhukar, 2013). The weights signify that some jobs may be more important than others, which means that the scheduler takes into account that delaying one job may be more "expensive" than delaying another one. Indeed, in cloud computing environments, some jobs may take priority over others because some jobs may serve more critical pipelines, inhibit further development progress, or belong to the most important clients (Shin et al., 2015).

Extensive research has been conducted on the non-capacitated counterparts of the problem. For instance, the Shortest Processing Time (SPT) priority rule minimizes the (non-weighted) sum of job completion times (Pinedo, 2012). An adapted version of the SPT, the Weighted Shortest Processing Time (WSPT) rule, is commonly used for the weighted version of the problem. Since the latter problem is $\mathcal{NP}$-complete for more than two machines (Garey and David, 1979), solution approaches have focused on developing polynomial-time heuristic approximation algorithms that bound the worst-case performance with respect to an optimal solution. An approximation algorithm A for a minimization problem P would have an approximation ratio ρ 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. These algorithms play a crucial role in resource allocation, scheduling, and production settings, as noted by Wang and Wenli (2021) and Gafarov et al. (2014). It has been proven by Eastman et al. (1964) that scheduling according to the WSPT priority rule provides a constant approximation ratio. Kawaguchi and Seiki (1986) found the WSPT approximation ratio (ρ) to be $(1 + \sqrt{2})/2$ and proved that it is tight.

In the capacitated setting, Im et al. (2016) were the first to develop a constant approximation algorithm for minimizing the *non-weighted* sum of completion times. They combined the Smallest Volume First (SVF) and the SPT priority rules. Their work, which was limited to the unweighted case, cannot be extended to the weighted case by simply adding weights to their analysis. Our work closes this gap by suggesting an efficient scheduling algorithm with provable theoretical bounds and good performance. Our algorithms are based on Weighted Smallest Volume First (WSVF) priority rule, where jobs are arranged in a non-decreasing order of the ratio between their processing time multiplied by the resource requirement (i.e., demand) and their weight. The algorithms we propose are suitable for both the weighted and non-weighted cases and can easily be implemented in real production and cloud computing systems.

The primary research contributions of this paper are:

(1) From a theoretical point of view, our research develops the first polynomial-time scheduling algorithm with a constant approximation ratio for the capacitated weighted completion time minimization problem.
(2) An important part of our analysis leads to $\left(1+\frac{1}{1-\alpha}\right)$-approximation algorithm, where α is equal to the maximum resource demand of any job. This analysis, which is the first one for the weighted case, also improves the best known approximation ratio for the *non-weighted* problem. In Im et al. (2016), the authors proved a $\left(\frac{3\alpha}{1-\alpha}+3\right)$-approximation ratio.
(3) From a modeling point of view, we develop a Mixed-Integer Linear Programming (MILP) formulation of the problem that is used for solving small problem instances for benchmarking purposes.
(4) From a practical standpoint, we show that the suggested algorithms can easily be used for scheduling offline (static) and online problems. Our experiments indicate that the algorithm's performance may be close to optimal for small problem instances and favorable with respect to common solution methods for real-world size problems.

The rest of the paper is organized as follows. Section 2 reviews the relevant literature. In Sect. 3, we formally define the problem and the MILP of this problem. In Sect. 4, we present various algorithms: the WSVF algorithm (Sect. 4.1), the Hybrid-WSVF (Sect. 4.2) and two versions of the WSVF algorithm for online environments in which jobs arrive over time (Sect. 4.3). In Sect. 5, we prove that the algorithms are implementable in polynomial time and bound the approximation ratio of the offline algorithm. Section 6 describes a thorough experimental study and analysis for various offline and online scenarios of this problem. The last section concludes the paper and suggests future research directions.

2 Literature review

The machine scheduling domain has been widely researched because of its theoretical and practical significance. In machine scheduling, as in many resource allocation settings such as queueing systems [e.g., Bitton et al. (2019)] and resource-constrained scheduling problems [e.g., (Hart et al., 2005; Balouka and Izack, 2021; Cohen et al., 2023)], the common assumption is that a resource/server (e.g., a machine) deals with a single job at any given time. We refer the interested reader to Kress et al. (2018) and Liu (2020) for more information about machine scheduling; here, we focus on recent contributions to the field of capacitated machine scheduling and closely related research.

Two types of problems that share some characteristics with the capacitated machine problem are the multiprocessor job scheduling problem (Bianco et al. (1995); Chen and Chung-Yee (1999)) and the consecutive multiprocessor job scheduling problem (Bukchin et al. (2020)). In these problems, it is assumed that machines can process, at most, one job at any time but a job can be processed by multiple machines simultaneously. In the consecutive version of the problem, a job can be processed only by a sequence of consecutive machines. Umang et al. (2013) considered the hybrid berth allocation problem that is motivated by the need to allocate berth space for vessels in container terminals. In the hybrid berth allocation problem setting, the quay is partitioned into a set of sections. One vessel can occupy more than one section at a time, and more than one vessel may share the same section at the same time. This setting is different than the capacitated machine setting since in the latter setting, one vessel (job) can be served by more than one section (machine). We note that the scale of a grid computing system (measured in the number of jobs and machines) is much larger than the scale of a port system.

Differently from the previous types of problems, under a capacitated machine setting, servers can process several jobs in parallel, up to their capacities.

Research about capacitated machine scheduling can be classified into offline settings under which the list of jobs and their properties are known, and online problems in which a stream of arriving jobs has to be scheduled.

Research about the offline capacitated scheduling problem typically aims to develop a scheduling algorithm with some performance guarantees. Some researchers aim at minimizing the processing makespan—that is, the completion time of the last job [e.g., Bougeret et al. (2011), Muter (2020), Jansen and Malin (2019) and Jansen and Trystram (2016)]. When jobs are associated with release times and deadlines, a natural objective function is to maximize the total weight of jobs completed before their deadlines (see Albagli-Kim et al. (2014) and Guo and Hong (2017)).

Im et al. (2016) developed a constant approximation algorithm for minimizing the sum of job completion times, without considering the more general weighted version of the problem. We consider a setting in which some jobs may be more important than others, thus, we develop approximation algorithms that solve the weighted version of the problem. These algorithms improve the approximation ratio for the non-weighted version that was presented in Im et al. (2016), and extend the work of Cohen et al. (2021).

Another line of research models the capacitated machine scheduling problem as an online problem in which jobs arrive over time (Kumar et al., 2019). Finding constant approximation algorithms for the online setting is an open problem. Many researchers have developed clever algorithms. Hota et al. (2019) developed an algorithm to maximize the throughput of jobs and minimize the response time (i.e., the time between a job's arrival and its scheduling). Other researchers who examined the problem from a different angle, focusing on the weighted sum of flow times or completion times, are Fox and Madhukar (2013); Im et al. (2016); Liu et al. (2019). Fox and Madhukar (2013) considered an online problem of weighted flow time minimization, assuming that jobs can be preempted with no penalty and delay. Such an assumption, however, may have important ramifications. Preemption may incur significant switching costs (e.g., setup costs) and memory loss as well as be forbidden due to system restrictions or client commitments.

Liu et al. (2019), 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 used Online Convex Optimization (OCO) to solve the scheduling optimization problem. Such 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. In this research, we suggest WSVF-based algorithms for the online setting of the capacitated parallel machine scheduling problem and evaluate their performance.

3 Problem definition and models

3.1 Problem definition and notation

We consider a set of N jobs, denoted as $\mathcal{N} = \{1, 2, ..., N\}$, that need to be processed on M identical machines, denoted as $\mathcal{M} = \{1, 2, ..., M\}$. Each job $j \in \mathcal{N}$ has a processing time p_j, demand d_j, and weight w_j, which are known in advance (this assumption will change when discussing online problems). Without loss of generality (w.l.o.g.), we assume that $p_j \in \mathbb{N}_+$ (by normalizing the values) and $d_j \in (0, 1]$ represents a fraction of the required demand with respect to a machine's capacity. Our focus is on a non-preemptive schedule, which means that once a job starts, it is processed without interruption until completion. An algorithm, $\mathcal{A}$, assigns each job j a pair (m_j, s_j), where $m_j \in \mathcal{M}$ represents the machine on which job j is processed and s_j determines its start time on that machine. We denote $T = \lceil \sum_{j \in \mathcal{N}} p_j \rceil$ as an upper bound on processing completion time and, denote $\mathcal{T} = \{0, 1, ..., T\}$. Given a schedule, for a machine $i \in \mathcal{M}$ and $t \in \mathcal{T}$, we define $G^i(t)$, the set of jobs that are processed on machine i at time t, formally:

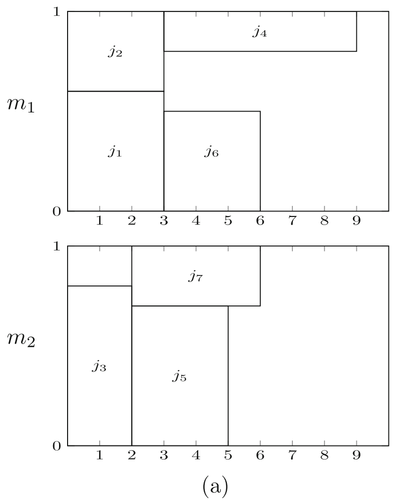
$$G^i(t) = \{j \in \mathcal{N} : m_j = i, t \in [s_j, c_j)\}.$$

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. Formally,

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

3.2 Problem illustration

Figure 1 illustrates the problem and suggests a feasible assignment. In this particular instance, there are two machines and seven jobs. Job parameters are provided in Fig. 1b. The assignment is shown in Fig. 1a, including the assignment of jobs to machines (m_j), and



(a)

jobs	p_j	d_j	w_j	m_j	s_j
1	3	0.6	9	1	0
2	3	0.4	4.8	1	0
3	2	0.8	4.8	2	0
4	6	0.2	2	1	3
5	3	0.7	3	2	2
6	3	0.5	2	1	3
7	4	0.3	1.5	2	2

(b)

Fig. 1 A feasible assignment for an instance of the capacitated parallel machine problem

start times (s_j). Note that, the width of a job represents its processing time and the height represents its demands as shown in Fig. 1a. Notably, in this assignment, $G^1(4) = \{4, 6\}$ and $G^2(3) = \{5, 7\}$. The weighted completion time is calculated as $\sum_{j=1}^{7}(p_j + s_j) \cdot w_j = (3 + 0) \cdot 9 + \cdots + (4 + 2) \cdot 1.5 = 105$.

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. We now define a MILP formulation for this problem.

3.3 The mixed-integer linear programming formulation

For the MILP formulation, we introduce binary decision variables, $s_{j,i,t}$, that are set to 1 if job j is assigned to machine i at start time t (i.e., when $m_j = i$ and $s_j = t$), and 0 otherwise. By utilizing these variables, we establish a set of constraints that ensure the feasibility of the assignment, as well as define the objective function's value:

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

subject to:

$$t \cdot s_{j,i,t} + p_j \leq c_j \qquad \forall j \in \mathcal{N}, i \in \mathcal{M}, t \in \mathcal{T} \tag{2}$$

$$\sum_{i \in \mathcal{M}} \sum_{t \in \mathcal{T}} s_{j,i,t} = 1 \qquad \forall j \in \mathcal{N} \tag{3}$$

$$\sum_{j \in \mathcal{N}} d_j \cdot \sum_{v=t-p_j+1}^{t} s_{j,i,v} \leq 1 \qquad \forall t \in \mathcal{T}, i \in \mathcal{M}$$

$$s_{j,i,t} \in \{0, 1\}, c_j \geq 0 \qquad \forall j \in \mathcal{N}, i \in \mathcal{M}, t \in \mathcal{T}. \tag{4}$$

The objective function aims to minimize the weighted sum of job completion times. Constraint (2) is used to bound the completion time c_j of each job j based on its starting and processing times. Constraint (3) ensures that each job starts exactly once. Constraint (4) guarantees that the (identical) machine capacities, normalized to 1, are satisfied by summing the demands of in-process jobs on each machine i. To enhance computational efficiency, we employ standard techniques to compute a tighter upper bound on T, which represents the maximum processing time.

The problem involves a total of $N \cdot M \cdot T$ variables and $N \cdot M \cdot T + N + M \cdot T$ constraints. It is worth noting that the number of variables and constraints has an exponential impact on the computational complexity of the problem.

4 The algorithms

In this section, we present the offline and online algorithms for the weighted completion time problem. Our algorithms follow a list-based approach, where jobs are ordered in a list and the algorithms iterate through the list to determine the earliest feasible time t for assigning the next job. To begin, we introduce the procedure *IsFeasible*, which determines whether it is feasible to assign a job with processing time p_j and demand d_j to machine i at time step t, given the current set of assigned jobs. If the assignment is feasible, the procedure returns True. Formally,

```
**Procedure** $IsFeasible(i, t, p_j, d_j)$
if $d_j + \sum_{h \in G_i(t')} d_h \leq 1$ for all $t' \in [t, t + p_j)$ then
    return True
else
    return False
end if
```

In this section we present the algorithms and in the next section (Sect. 5), we analyze them to establish approximation bounds.

We also present two algorithms specifically designed for an online setting, where jobs enter the system over time without prior knowledge of their arrival time, duration, and demand. Both algorithms utilize the WSVF priority rule for sorting jobs. The first one employs batch dispatch, while the second algorithm utilizes continuous dispatch of the jobs. We present and provide more details about the algorithms below.

4.1 WSVF algorithm

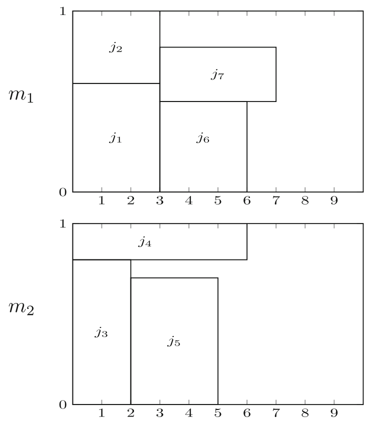
The WSVF algorithm sorts the jobs in a non-decreasing order based on their volume over weight values, i.e., v_j/w_j, where $v_j = p_j \cdot d_j$ is job j's volume. The unassigned job with the highest priority (i.e., the one with the smallest WSVF value) is scheduled on the earliest available machine at time t where it is processed until completion. If multiple machines are available at the same earliest time, WSVF assigns the job to the machine with the lowest index, filling one machine before moving to the next.[1] The formal description of the WSVF algorithm is as follows:

```
Algorithm 1: WSVF
    Set $\mathcal{J}$ the list of jobs
    Sort $\mathcal{J}$ according to $(p_j \cdot d_j)/w_j$
    for job $j \in \mathcal{J}$ do
        Find the earliest time $t^*$ such that exists $i^* \in \mathcal{M}$ where IsFeasible($i^*$,$t^*$,$p_j$,$d_j$)
        Set $m_j = i^*$ and $s_j = t^*$
    end for
```

By definition, the assignment produced by Algorithm 1 is guaranteed to be feasible. In Sect. 5.1, we demonstrate that the approximation ratio of the algorithm depends on $\alpha = \max_{j \in \mathcal{N}} d_j$, which represents the maximum resource demand among all jobs.

An example of the WSVF algorithm Fig. 2, demonstrates how WSVF works on the example shown in Fig. 1. Note that the jobs are sorted in a non-decreasing order of $p_j \cdot d_j/w_j$ (e.g., $p_1 \cdot d_1/w_1 = 0.2$, and $p_2 \cdot d_2/w_2 = 0.25$). The jobs are assigned based on their priority order, with each job scheduled on the machine with the earliest available capacity for p_j time steps. As shown in the figure, jobs j_1, j_2, j_3, and j_4 are scheduled at time 0 according to the available capacities. Job j_5 can only be scheduled after the completion of j_3. Therefore,

[1] In Algorithm 1, ties are broken randomly as the tie-breaking procedure does not impact the bound. In the computational study, we report on the performance of different tie-breaking procedures; see Sect. 6.



(a)

jobs	p_j	d_j	w_j	$\frac{p_j \cdot d_j}{w_j}$	m_j	s_j
1	3	0.6	9	0.2	1	0
2	3	0.4	4.8	0.25	1	0
3	2	0.8	4.8	0.33	2	0
4	6	0.2	2	0.6	2	0
5	3	0.7	3	0.7	2	2
6	3	0.5	2	0.75	1	3
7	4	0.3	1.5	0.8	1	3

(b)

Fig. 2 An illustration of the WSVF algorithm's run

$s_5 = c_3 = 2$, and jobs j_6 and j_7 await the completion of jobs j_1 and j_2. The objective function value in the example is $\sum_{j=1}^{8} w_j \cdot c_j = 3 \cdot 9 + 3 \cdot 4.8 + \cdots + 7 \cdot 1.5 = 100.5$.

4.2 Hybrid-WSVF

We now introduce the Hybrid-WSVF algorithm, which eliminates the dependence on α and provides a constant approximation ratio for any instance with $M \geq 2$. In the appendix, we also present an algorithm with a constant approximation ratio for the case of $M = 1$. The algorithm for $M \geq 2$ involves splitting the set of jobs I into two subsets, I^l and I^h. A job is included in I^h if its resource demand is strictly greater than $1/2$. Similarly, the set of machines is split into two disjoint sets. The first set contains $M_1 = \lceil \frac{2(M-2)}{3} \rceil + 1$ machines and processes the jobs in I^l using the WSVF algorithm, while the second set contains $M_2 = M - M_1$ machines and processes the jobs in I^h using the WSPT algorithm.

The Hybrid-WSVF algorithm:

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 $M_2 = M - M_1$ machines using WSPT.
```

4.3 Online algorithms—continuous and batch dispatch

We demonstrate how we extend the WSVF algorithm to settings in which jobs arrive to the system over time without prior knowledge about their arrival time, duration and demand—also known as online problems. Fox and Madhukar (2013) proved that there is no constant approximation algorithm for the online weighted capacitated machine scheduling problem. We focus on developing online versions of the WSVF algorithm.

When utilizing WSVF in online scenarios, it becomes necessary to determine the time points for updating the job queue based on the priority rule and when to make scheduling decisions for each job. In contrast to offline settings where all decisions are made at time 0, online settings require ongoing decision-making as new jobs arrive.

We propose two distinct online algorithms. The first algorithm, referred to as *batch dispatch*, sorts the jobs that arrive within a predetermined time interval using the WSVF priority rule and schedules them on the available machines. The batch dispatch algorithm commits capacity in advance, thereby limiting the flexibility to accommodate jobs arriving in subsequent time intervals. This algorithm provides end-users with certainty regarding their job's completion time, which can be crucial in certain systems and for specific users.

The second algorithm, denoted as *continuous dispatch*, applies the WSVF priority rule whenever a new job enters the system. Unlike batch dispatch, it dispatches a job to a machine only if the job can begin execution immediately. Consequently, the continuous dispatch algorithm does not provide end-users with certainty regarding job completion times. It does, however, offer decision-makers greater flexibility in resource utilization, which is expected to result in improved objective function values compared to the batch dispatch algorithm.

Here is the formal definition of the batch dispatch algorithm:

Algorithm 3: Batch Dispatch WSVF

```
for $\hat{t} \in \mathcal{T}$ do
  Set $\mathcal{J}_{\hat{t}}$ the list of unassigned jobs that are available at time $\hat{t}$
  Sort $\mathcal{J}_{\hat{t}}$ according to $(p_j \cdot d_j)/w_j$
  for job $j \in \mathcal{J}_{\hat{t}}$ do
    Find the earliest time $t^* \geq \hat{t}$ such that exists $i^* \in \mathcal{M}$ such that $IsFeasible(i^*,t^*,p_j,d_j)$
    Set $m_j = i^*$ and $s_j = t^*$
  end for
end for
```

The formal definition of the continuous dispatch algorithm, which commits to the starting time of a job only if the current time step ($\hat{t}$) is available, is:

Algorithm 4: Continuous Dispatch WSVF

```
for $\hat{t} \in \mathcal{T}$ do
  Set $\mathcal{J}_{\hat{t}}$ the list of unassigned jobs that are available at time $\hat{t}$
  Sort $\mathcal{J}_{\hat{t}}$ according to $(p_j \cdot d_j)/w_j$
  for job $j \in \mathcal{J}_{\hat{t}}$ do
    If exists $i^* \in \mathcal{M}$ such that $IsFeasible(i^*,\hat{t},p_j,d_j)$
    Set $m_j = i^*$ and $s_j = \hat{t}$
  end for
end for
```

5 Analysis of the algorithms

First, we demonstrate that the above algorithms are implementable in poly-time in N, M (note that the bound on running time does not depend on p_j values).

Lemma 5.1 *Given an instance with N jobs and M machines, the algorithms can be implemented in $O(M \cdot N^3)$ time.*

Proof The key step in these algorithms is to find the earliest processing time t to assign the next job. Hence, we need to show that for each job, the earliest start time can be determined in polynomial time. We will present an implementation with $O(N^2 \cdot M)$ running time, resulting in a total running time of $O(N^3 \cdot M)$.

Let us say that for a specific machine, given an assignment of k jobs on that machine, the earliest possible starting time for a job occurs when the available demand changes. Under such an assignment, there are at most $k + 1 = O(N)$ time points where the total demand changes. By preserving the time points of each machine in a linked list, it is possible to determine the earliest processing time in $O(N^2)$ time. By iterating over at most $O(N)$ time points and determining for each point whether it is a feasible time, we can complete this step in at most $O(N)$ time. To find the earliest point among all machines, we simply identify the minimum starting time among all M machines.

Finally, sorting the jobs can be implemented in $O(N \log N)$ time. The Hybrid-WSVF algorithm utilizes the WSVF and WSPT algorithms on subsets of the machines. The WSPT algorithm can be implemented using the WSVF algorithm (with $d_j = 1$ for all jobs), making it polynomial-time implementable. The online algorithms mainly utilize the procedure for finding the earliest time among all machines, which, as we have demonstrated, is implementable in polynomial time.

Thus, all the above algorithms can be implemented in $O(M \cdot N^3)$ time. □

5.1 WSVF algorithm analysis

Next, we prove the approximation ratio of the WSVF algorithm:

Theorem 5.2 *Given an upper bound on the demand of a job $\alpha < 1$, then the WSVF algorithm is a $\left(\frac{1}{1-\alpha} + 1\right)$-approximation algorithm for the weighted completion time minimization problem.*

To complete the proof of Theorem 5.2, 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). To do this, we take the ordered set of jobs I, and for each job in the set, we fix the duration as $\hat{p}_j = p_j \cdot d_j$, which is the so-called volume of job j in the original instance. We then set the demand to $\hat{d}_j = 1$. We note that the compressed instance preserves the jobs' order 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 WSVF for instance I using M machines and by $C^*_M(I)$ the optimal cost of scheduling I on M machines.

We make 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 (Smith, 1956).

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 5.3 $C_N(I) \leq C^*_M(I)$.

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

Claim 5.4 $C^*_M(\hat{I}) \leq C^*_M(I)$.

Proof Consider, w.l.o.g., one of the machines M_i and let $j_1, j_2, ..., j_k$ be the jobs assigned to this machine according to the completion time order 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 cost smaller than or equal 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 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 our extending the volume preservation argument to M machines. □

We now construct a lower bound on $C^*_M(\hat{I})$ using a single machine optimal cost $C^*_1(\hat{I})$. We provide a different proof than the one by (Eastman et al., 1964), using the compressed instance definition.

Claim 5.5 $C^*_M(\hat{I}) \geq \frac{1}{M} C^*_1(\hat{I})$.

Proof We first mark $I_{\frac{1}{M}}$ as a transformed version of $\hat{I}$ such that we set the demand of each job to $1/M$ (instead of 1); the weights and processing times stay the same. We then claim that

$$C^*_M(\hat{I}) \geq C^*_1(I_{\frac{1}{M}}),$$

since the optimal solution of $\hat{I}$ over M machines is also feasible for scheduling $\hat{I}'$ over a single machine (using imaginary bounds in the size of $1/M$).
Additionally, we set $\hat{I}_{\frac{1}{M}}$ as the compressed version of $I_{\frac{1}{M}}$. By Claim 5.4, we have

$$C^*_1(I_{\frac{1}{M}}) \geq C^*_1(\hat{I}_{\frac{1}{M}}).$$

Finally, we note that

$$C^*_1(\hat{I}_{\frac{1}{M}}) = \frac{1}{M} C^*_1(\hat{I})$$

because $\hat{I}_{\frac{1}{M}}$ is a similar problem to $\hat{I}$ except that the duration of each job is shorter by a factor of $1/M$.
Therefore, it follows that

$$C^*_M(\hat{I}) \geq \frac{1}{M} C^*_1(\hat{I})$$

. □

We now compare $C_M(I)$ and $C_1(\hat{I})$ by bounding the start time s_j for each job $j \in N$. We prove that s_j can be bounded by a factor that depends on α 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 5.6 $\forall j \in \mathcal{N} : s_j \leq \frac{1}{(1-\alpha)M} \Sigma_{h=1}^{j-1} v_h$.

Proof To prove Lemma 5.6 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 Im et al. (2016) 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 $D^i_j(t) = \sum_{h \in G^i(t), h \leq j} d_h$, 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 $D^i_{j-1}(t) \geq 1 - \alpha$. □

Claim 5.7 For all $i \in \mathcal{M}$, $j \in \mathcal{N}$, $min\{D^i_j(t), 1-\alpha\}$ 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 (the jobs assigned to other machines do not affect this machine's demand D_i) with the same priority order as in I. We prove the claim using induction on j. First, consider the case of $j = 1$,

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

This 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 algorithm $D_j(t) = D_{j-1}(t) > 1-\alpha$, 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, $D_{j-1}(t) + \alpha \leq 1$), 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 algorithm would assign job j before time s_j.

We now look at the value $min\{D_j(t), 1-\alpha\}$ 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 together with the induction assumption, we can conclude that $min\{D_j(t), 1-\alpha\}$ is non-increasing with t. □

Claim 5.8 For all $i \in \mathcal{M}$, $j \in \mathcal{N}$ and $0 \leq t < s_j$, we have $D^i_{j-1}(t) > 1-\alpha$.

Proof We follow the proof of Claim 5.7: if at any $t < s_j$, $D_{j-1}(t) \leq 1-\alpha$, then by the invariant of Claim 5.7 and its definition, WSVF would process job j earlier than s_j. □

Next, we use Corollary 5.8 to set an upper bound on the starting time of every job j and to conclude the proof of Lemma 5.6. By Corollary 5.8, to process the first $j-1$ jobs, each machine uses at least $(1-\alpha)$ of its capacity until time s_j. 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:

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

Using the above results, we now prove the main lemma for bounding $C_M(I)$:

Lemma 5.9 *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 By WSVF, for every job $j \in \mathcal{N}$,

$$c_j = p_j + s_j \leq p_j + \frac{1}{(1-\alpha)M} \cdot \sum_{h=1}^{j-1} v_h = p_j + \frac{1}{(1-\alpha)M} \cdot \sum_{h=1}^{j-1} \hat{p}_h, \tag{5}$$

where the LHS equality exists by definition, the inequality follows from Lemma 5.6, 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 Eq. 5 and the second inequality holds since $C_N(I) = \sum_{j=1}^{n} w_j \cdot p_j$, and $C_1(\hat{I}) = \sum_{j=1}^{n} (w_j \cdot \sum_{h=1}^{j} \hat{p}_h) \geq \sum_{j=1}^{n} (w_j \cdot \sum_{h=1}^{j-1} \hat{p}_h)$. □

Finally, we prove Theorem 5.2, which follows immediately from Lemma 5.9 and Claims 5.4 and 5.5.

Proof of Theorem 5.2

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

From the above, we can deduce that for high loaded systems ($C_N(I) << C_M^*(I)$) with low demand jobs ($\alpha << 1$), WSVF approaches the optimal solution. □

5.2 Hybrid-WSVF algorithm analysis

Our analysis for the Hybrid-WSVF algorithm relies on a number of observations. First, 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; therefore, scheduling WSPT for this set provides a constant approximation ratio. Second, WSVF provides a constant approximation ratio on jobs with demands equal to or smaller than $1/2$. Moreover, both algorithms are robust. Scheduling using these algorithms on a constant fraction of the machines would affect the objective only by a constant factor. Thus, by splitting the jobs based on their resource demands over two machines, we achieve a constant approximation algorithm.

Theorem 5.10 *HYBRID-WSVF is a $4+o(\frac{1}{M})$-approximation algorithm of 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. We have,

$$\begin{aligned} C_{M_1}(I^l) &\leq C_N(I^l) + \frac{1}{(1-\alpha)M_1} \cdot C_1(\hat{I}^l) \\ &\leq C_N(I^l) + \frac{2}{M_1} \cdot C_1(\hat{I}^l) \\ &= C_N(I^l) + \frac{2M}{M_1} \cdot \frac{C_1(\hat{I}^l)}{M} \\ &\leq C_M^*(I^l) + 2 \cdot \frac{M}{M_1} C_M^*(I^l), \end{aligned}$$

where the first inequality is by Lemma 5.9, and the second inequality follows since the low-demand jobs the values are within the interval $(0, 0.5]$; therefore, we set $\alpha = 0.5$ and the last inequality follows from Observation 5.3 and Claims 5.4 and 5.5.

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 Eastman et al. (1964) who proved the following bound for 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 5.3 and Claim 5.5.

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 $C^*_M(I) \geq C^*_M(I^h) + C^*_M(I^l)$, for $M \geq 2$, we have:

$$
\begin{aligned}
C_M(I) &= C_{M_1}(I^l) + C_{M_2}(I^h) \\
&\leq C^*_M(I^l) + 2 \cdot \frac{M}{M_1} C^*_M(I^l) + C^*_M(I^h) + \frac{M}{M_2} C^*_M(I^h) \\
&\leq C^*_M(I^l) + \left(3 + \frac{3}{M-1}\right) C^*_M(I^l) + C^*_M(I^h) + \left(3 + \frac{3}{M-1}\right) C^*_M(I^h) \\
&\leq \left(4 + \frac{3}{M-1}\right) (C^*_M(I^l) + C^*_M(I^h)) \\
&\leq \left(4 + \frac{3}{M-1}\right) C^*_M(I),
\end{aligned}
$$

which concludes the proof for Theorem 5.10. □

6 A computational study

We conducted a computational study to complement our theoretical results and specifically to provide insights about:

1. The performance gap between WSVF and the optimal solution (Sect. 6.1).
2. The average performance of WSVF compared to other common algorithms in offline (Sect. 6.2) and online (Sect. 6.3) settings.

We note, in passing, that the Hybrid-WSVF algorithm is important since its design enables us to compute a theoretical bound. For the computational experiments, we expect the WSVF algorithm to be better on average, thus we use it.

The following is a brief outline of the study. First, we use a MILP formulation of the weighted completion time minimization problem for solving relatively small instances of the capacitated parallel machine problem, with up to 10 machines and 50 jobs. This formulation enables comparing optimal solutions to those generated by WSVF. In the second part of the study, we scale up the instance sizes in favor of comparing WSVF pwith other known heuristic algorithms for realistic scenarios. For this, we use problems with up to 80 machines and 30000 jobs. The third part of the study focuses on testing the performance of WSVF and some alternatives for online settings in which jobs arrive over time. We test scenarios in which job processing times are deterministic or stochastic with up to 10 machines and 4000 jobs.

We note that for cases where more than one machine is available at the same earliest time, we tested three different tie-breaking methods. Best-fit, First-index-fit, and Random. First-index-fit always picks the machine with the smallest index and Best-fit picks the smallest free partition that meets the demand of the requesting job. Finally, Random selects a candidate machine randomly. Preliminary tests indicated that for our setting, Best-fit performed best; thus, we present its results.

6.1 Comparing WSVF performance to the optimal solution

Section 3.3 detailed a MILP formulation of a minimization problem that can be solved to optimality for small scenarios and compared to corresponding WSVF solutions, which is what we present next.

For this, we designed a variety of scenarios characterized by the number of jobs, the number of machines, and the maximum demand of a job (i.e., α). We investigated $\alpha \leq 0.5$ since when d_j values are larger than 0.5, a machine can only process, concurrently, a single job. The details about the range of scenarios can be found in Table 1a. The parameters for the tested scenarios were generated as detailed in Table 1b.

Overall, we collected statistics for 60 generated scenarios, where each one was replicated 50 times and solved via WSVF and by solving the MILP formulation in Problem P to optimality. To attain optimal solutions, we used the Gurobi solver (Gurobi, 2021) and Python's 'mip' package (https://www.python-mip.com). WSVF solutions were found using a Python-based simulator that we developed for this purpose.

As can be seen in Fig. 3, the maximal gap between WSVF and and the optimal solutions is 6%. Additionally, we have witnessed that the ratio between the optimal solution to the WSVF solution remains stable across scenarios of different scales and across varying loads on the machines (e.g., the number of jobs per machine) but changes with α. A change of α also affects the average of the jobs' demands $\bar{d}$ since $d_j \sim U(0.05, \alpha)$. Figure 3 demonstrates that the ratio increases with α.

Table 1 Experimental settings (1a) describes the controlled parameters—the numbers of jobs and machines, and (1b) how the weights, processing times, and demands were generated across scenarios

(a)		
N	M	α
10–50	2–10	0.05-–0.5
(b)		
w_j	$U(1, 10)$	
p_j	$U(1, 6)$	
d_j	$U(0.05, \alpha)$	

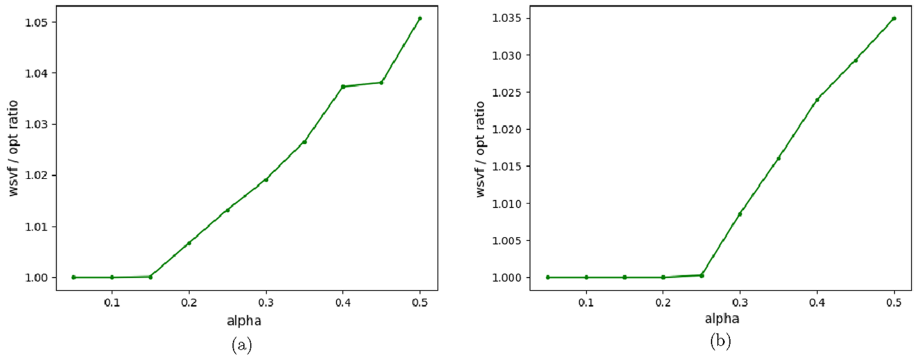


Fig. 3 The ratio between WSVF and the optimal solution as a function of α for two different scenario sets: Scenario Set **a** consists of 21 jobs scheduled over three machines, and Scenario Set **b** consists of 20 jobs scheduled over four machines

6.2 WSVF versus common heuristic algorithms

We developed a Python-based simulator to test WSVF performance for realistic-size scenarios following the work of Im et al. (2016) with synthetic data (Table 2a characterizes the scenarios) and using real-world scenarios from a partner organization.

Real-world scenarios were designed following our collaboration with a partner organization, which is a well-known international technology corporation (unfortunately, we cannot disclose identifying details). The organization schedules multiple jobs on its grid-computing system for developing and testing products in a setting that perfectly matches the capacitated parallel machine scheduling problem. The scenarios included varying numbers of jobs and machines that reflect different loads within the original real-world system (see details in Table 3). Job characteristics for weights, processing durations and required demands, as replicated from the real-world system, can be described by the triplets (minimum, mean, maximum). We note, in passing, that grid computing environments may include larger numbers of machines but they are typically divided into pools of machines about the sizes of the pools used in our experiments. On any case, the algorithms that we propose are polynomial in the size of the input and thus can be used for larger scenarios.

WSVF performances were compared with the following alternatives from previous capacitated machine scheduling studies: The SPT algorithm that prioritizes the jobs with the shortest processing times, WSPT that also considers the job weight, and SVF that prioritizes jobs with smaller $p_j \cdot d_j$ values. Finally, Random creates a random prioritization order.

WSVF exhibits favorable performance over its alternatives across the synthetic and real data sets. Its advantage over the alternative algorithms increases with the increased load (i.e., a higher number of jobs per machine) as can observed from the figures.

In agreement with the theoretical analysis, the experiments show that the gap between WSVF and the optimal solution, as manifested by the WSVF-to-optimal-solution ratio, decreases as α gets smaller. The intuition is that as job demands increase, it becomes more

Table 2 Setup of experiments with synthetic data. (a) describes the intervals for the numbers of jobs and machines, and (b) describes how weights, processing times and demands were generated

(a)	
N	M
3000–16,000	20–80

(b)	
w_j	$U(0.1,5)$
p_j	$70\% : U(1, 100), 15\% : U(300, 350), 15\% : U(450, 500)$
d_j	$75\% : U(0.05, 0.5), 25\% : U(0.5, 1)$

Table 3 Setup of experiments with real-world data. (a) describes the intervals for the numbers of jobs and machines, and (b) describes how weights, processing times and demands were generated

(a)	
N	M
15,000–30,000	20–80

(b)	
w_j	(0.05, 7, 15)
p_j	(1, 600, 7000)
d_j	(0.002, 0.02, 0.9)

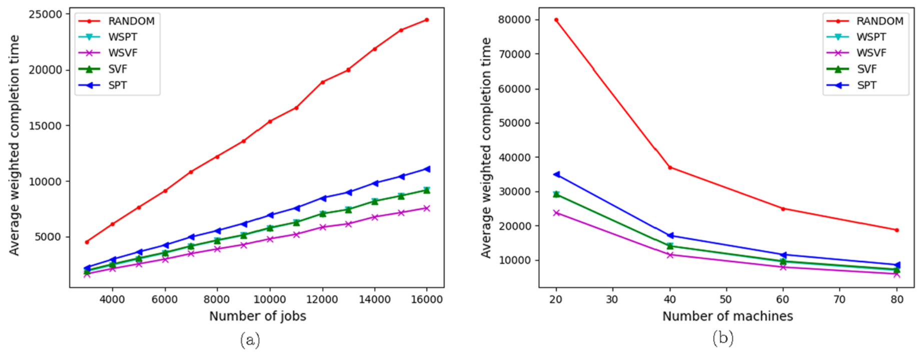


Fig. 4 The objective function values yielded by different algorithms on synthetic data: **a** the results for 50 machines as a function of the number of jobs, and **b** the results for 20, 000 jobs as a function of the number of machines

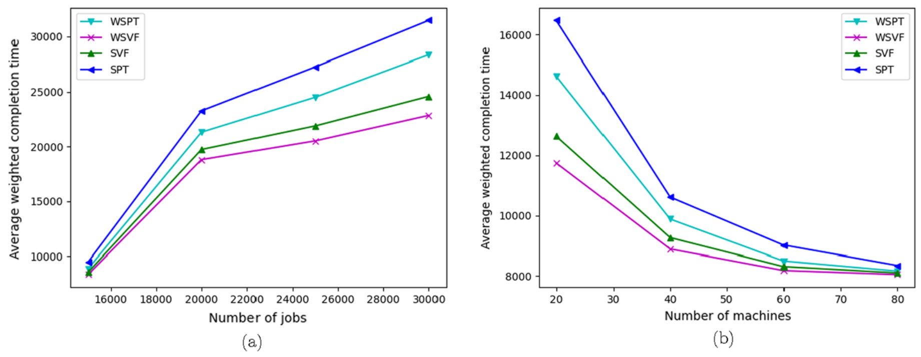


Fig. 5 The objective function values yielded by different algorithms on real data: **a** the average weighted completion time for a setting with 50 machines as a function of the number of jobs, and **b** the average weighted completion time for 15, 000 jobs as a function of the number of machines

complicated to pack sets of jobs on each machine. Hence, we expect more fragmentation of jobs in non-optimal scheduling (Figs. 4, 5).

6.3 Online settings

We now evaluate the performance of the online Algorithms 3 and 4 that use batch- and continuous-dispatch, respectively. We conducted experiments using a Python-based simulator that simulates online grid computing environments. The focus is on settings in which jobs, arriving randomly to the system, are assumed to have deterministic durations (Sect. 6.3.1)—for example., based on their observed class and a nondeterministic duration that can be estimated (e.g., by the job's owner or a prediction algorithm) as customary in several cloud computing systems (Sect. 6.3.2).

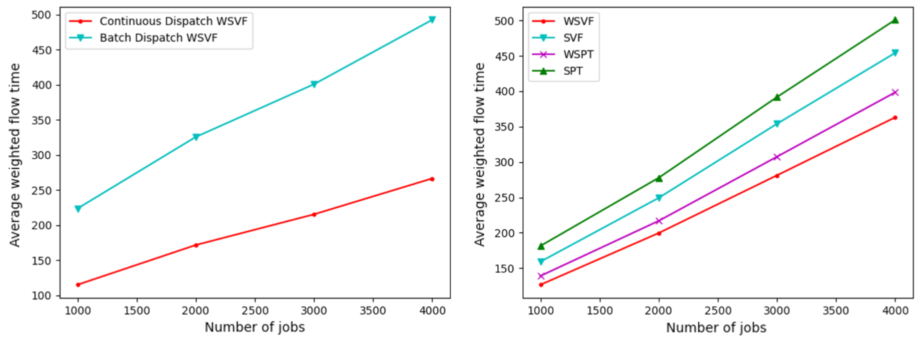
6.3.1 Deterministic processing times

For the first experiment, the jobs' arrival time is random but an observed arriving job is assumed to have a deterministic duration (an assumption that we relax in the next experiment).

Table 4 Online deterministic experimental setup. (4a) describes the numbers of jobs and machines that were tested and (4b) how the weights, processing times, release times, and demands were generated

(a)	
N	M
1000–4000	10

(b)	
w_j	$U(1, 10)$
p_j	$U(1, 6)$
r_j	$U(1, 30)$
d_j	$U(0.1, 0.5)$



(a) Continuous vs. batch dispatch WSVF (b) Continuous dispatch WSVF vs. alternatives

Fig. 6 The average flow time using the online WSVF algorithm and other alternatives as a function of the number of jobs

The arrival time values, r_j and all other jobs' parameters are set as detailed in Table 4b. We compared the performance of the two algorithms as a function of the number of jobs that need to be scheduled on 10 machines.

The results, in Fig. 6a, demonstrate that the cost (i.e., objective function value) of providing the end-user with full certainty about their job's completion time by using the batch dispatch method is approximately double the cost of the continuous dispatch method. Experiments with other parameter combinations exhibit similar results. Using the same experimental setup, we evaluated the continuous dispatch WSVF with common heuristics—namely, the SPT, WSPT, and SVF algorithms. WSVF performs consistently better than the alternatives for an online setting as can be seen in Fig. 6b.

6.3.2 Non-deterministic processing times

This experiment emulates a setting in which the processing times of the arriving jobs are unknown and can only be estimated. Hence, we evaluate how different algorithms perform in a setting where scheduling decisions are based on estimations of the actual processing times and what happens when the accuracy of these estimations changes. Accordingly, we present two experiments in which the realized processing durations (p_j) may be different than their estimations ($\hat{p}_j$).

In the first experiment (Fig. 7a), processing time estimations are drawn from a normal distribution, $\hat{p}_j \sim N(p_j, \sigma)$, where we control σ and change it to check the performance under varying estimation accuracy.

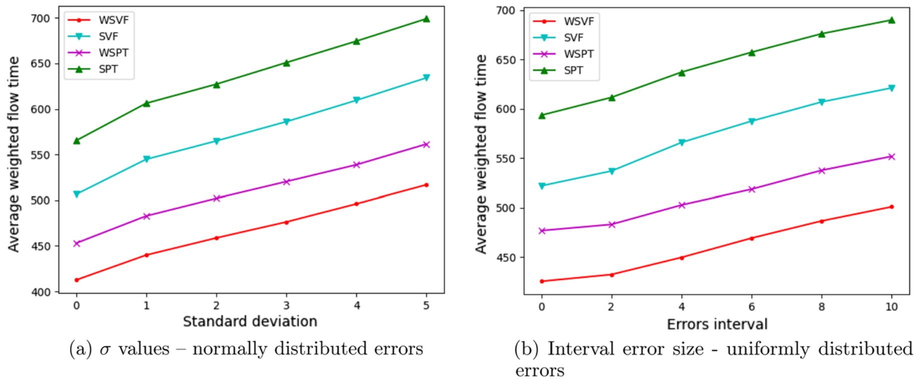
Table 5 Online non-deterministic experimental setup. (5a) describes the the numbers of jobs and machines, and the accuracy of estimations represented by σ and I), and (5b) describes how weights, processing- and release-times, and demands were generated

(a)

N	M	σ	I
2000	8	$0-5$	$0-10$

(b)

w_j	$U(1, 10)$
p_j	$U(1, 6)$
r_j	$U(1, 30)$
d_j	$U(0.1, 0.5)$



(a) σ values – normally distributed errors (b) Interval error size - uniformly distributed errors

Fig. 7 Continuous dispatch WSVF and alternatives as a function of processing estimation accuracy

Similarly, in the second experiment (Fig. 7b), processing time estimations are generated from a uniform distribution, $\hat{p}_j \sim U(p_j - 0.5 \cdot I, p_j + 0.5 \cdot I)$ where the interval size, I, is changed. Details about the different scenarios and parameter settings can be found in Tables 5a and 5b.

In both experiments, WSVF provides the best results and the ratios between the results of the different algorithms are stable. Hence, we can infer that even though we prove a constant approximation ratio only in offline deterministic environments, WSVF can be efficient even in a nondeterministic online environment such as the one demonstrated in the experiments above.

7 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), we are not familiar with constant approximation algorithms for the related problem of minimizing the weighted sum of job completion times. This paper closes this gap by suggesting an algorithm that also improves the best-known approximation ratio for the non-weighted problem (for $M \geq 2$). We show that the WSVF algorithm guarantees a constant approximation ratio in the worst case and, equally important, performs, on average, within 6% of the optimal solution for scenarios with up to 50 jobs and 10 machines and demonstrates superior performance with respect to other common methods. We also showed that although the approximation ratio was proved only for offline-deterministic environments, the algorithm, with small adaptations, can be applied in online environments.

We suggest three possible future research directions. The first significant theoretical extension is to consider problems with multiple capacitated resources. In real-world cloud computing environments, CPU, memory, storage, and bandwidth may be scarce resources. Such research could start by modeling a multidimensional resource requirement for each job. A second extension would be to find performance guarantees in stochastic environments in which, for example, processing durations can be characterized in probabilistic terms. The third research direction would be to develop a model that accommodates jobs with release dates and deadlines. Such research may be considered to be a natural extension of the current model.

Acknowledgements The research of I.C. was funded by ISF grant 226/21.

Data availability The research data and the code for this article can be found in the following GitHub repository: https://github.com/iyarzaks/scheduling_simulator

Declaration

Human participants This research does not contain any studies with human participants performed by any of the authors.

Conflict of interest Ilan Reuven Cohen declares that he has no conflict of interest. Izack Cohen declares that he has no conflict of interest. Iyar Zaks declares that he has no conflict of interest.

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 problem for capacitated machines, there exists a constant competitive algorithm for any number of machines. Note that Algorithm 2 is well-defined for $M \geq 2$ machines; for $M = 1$ machine, the partition approach cannot be applied. Instead, we extend the ideas of the non-weighted case for $M = 1$ in Im et al. (2016) 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 as having width p_j and height d_j.

The algorithm depends on $\epsilon > 0$. Since we are only interested in a constant approximation and a polytime algorithm, we can just set $\epsilon = 0.1$. The value of ϵ affects the running time of the knapsack algorithm, and the running time is polynomial for a constant ϵ. Therefore, the running time of the algorithm is polynomial. The algorithm works iteratively until it assigns all jobs, where in iteration ℓ, the algorithm will execute the following steps:

(1) Solve a knapsack packing problem: Compute J_ℓ, a maximum weighted set of jobs with a total volume less than 2^ℓ using a $1 + \epsilon$ resource augmentation knapsack algorithm.
(2) Pack the jobs in J_ℓ 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 scheduled more than once when using the above algorithm. Of course, this is unnecessary, and in practice, the job will be scheduled only according to the first strip in which it appears. 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, 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. For this problem, we apply the resource augmentation solution of Williamson and David (2011), who proved that given a bound on the total size L, there exists a polynomial time in which a set with a 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 Jansen and Zhang's algorithm (Jansen and Guochuan, 2007), which packs rectangles (without rotations) with a total volume of L and maximal height of 1 into 3 strips of width L and height 1.

Algorithm 5: 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 $[3(1+\epsilon)\sum_{h=0}^{\ell-1} 2^h, 3(1+\epsilon)\sum_{h=0}^{\ell} 2^h)]$
end for
```

Theorem 7.1 *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 2D packing solution strip 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 ℓ. Note that jobs in J_ℓ are scheduled until time $3(1+\epsilon)\cdot(2^{\ell+1}-1)$. In addition, when jobs in J_ℓ 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 Algorithm 5 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_ℓ up to time 2^ℓ, because by volume preservation, it is impossible to pack more than 2^ℓ volume by this point and, by the correctness of the knapsack algorithm, this is the maximum profit (weight) that can be packed by W_ℓ if the total volume is at most 2^ℓ; therefore, there are jobs with a total weight of at least $W - W_\ell$ that are unprocessed at time 2^ℓ. 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 above two inequalities, we have that PackAndSchedule is a $12(1+\epsilon)$ approximation algorithm as required. □

References

Albagli-Kim, S., Shachnai, H., & Tamir, T. (2014). Scheduling jobs with dwindling resource requirements in clouds. In *IEEE INFOCOM 2014-IEEE conference on computer communications* (pp. 601–609). IEEE.

Balouka, N., & Cohen, I. (2021). A robust optimization approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research, 291*(2), 457–470.

Bianco, L., Blazewicz, J., Dell'Olmo, P., & Drozdowski, M. (1995). Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors. *Annals of Operations Research, 58*(7), 493–517.

Bitton, S., Cohen, I., & Cohen, M. (2019). Joint repair sourcing and stocking policies for repairables using Erlang-A and Erlang-B queueing models. *IISE Transactions, 51*(10), 1151–1166.

Bougeret, M., Dutot, P.-F., Jansen, K., Robenek, C., & Trystram, D. (2011). Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms. *Discrete Mathematics, Algorithms and Applications, 3*(04), 553–586.

Bukchin, Y., Raviv, T., & Zaides, I. (2020). The consecutive multiprocessor job scheduling problem. *European Journal of Operational Research, 284*(2), 427–438.

Chen, J., & Lee, C.-Y. (1999). General multiprocessor task scheduling. *Naval Research Logistics (NRL), 46*(1), 57–74.

Cohen, I. R., Cohen, I., & Zaks, I. (2021). Weighted completion time minimization for capacitated parallel machines. In *International workshop on approximation and online algorithms* (pp. 130–143). Springer.

Cohen, I., Postek, K., & Shtern, S. (2023). An adaptive robust optimization model for parallel machine scheduling. *European Journal of Operational Research, 306*(1), 83–104.

Damodaran, P., Ghrayeb, O., & Guttikonda, M. C. (2013). GRASP to minimize makespan for a capacitated batch-processing machine. *The International Journal of Advanced Manufacturing Technology, 68*(1–4), 407–414.

Eastman, W. L., Even, S., & Martin Isaacs, I. (1964). Bounds for the optimal scheduling of n jobs on m processors. *Management Science, 11*(2), 268–279.

Fox, K., & Korupolu, M. (2013). Weighted flowtime on capacitated machines. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on discrete algorithms* (pp. 129–143). SIAM.

Gafarov, E. R., Dolgui, A., & Werner, F. (2014). A new graphical approach for solving single-machine scheduling problems approximately. *International Journal of Production Research, 52*(13), 3762–3777.

Garey, M. R, & Johnson, D. S. (1979). Computers and intractability. In *A guide to the theory of NP-completeness*.

Guo, L., & Shen, H. (2017). Efficient approximation algorithms for the bounded flexible scheduling problem in clouds. *IEEE Transactions on Parallel and Distributed Systems, 28*(12), 3511–3520.

Gurobi Optimization, LLC. (2021). Gurobi optimizer reference manual. https://www.gurobi.com

Hart, E., Ross, P., & Corne, D. (2005). Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines, 6*(2), 191–220.

Hota, A., Mohapatra, S., & Mohanty, S. (2019). Survey of different load balancing approach-based algorithms in cloud computing: A comprehensive review. In: *Computational intelligence in data mining* (pp. 99–110).

Im, S., Naghshnejad, M., & Singhal, M. (2016). 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.

Jain, N., & Choudhary, S. (2016). Overview of virtualization in cloud computing. In *2016 Symposium on colossal data analysis and networking (CDAN)* (pp. 1–4). IEEE.

Jansen, K., & Rau, M. (2019). Linear time algorithms for multiple cluster scheduling and multiple strip packing. In *European conference on parallel processing* (pp. 103–116). Springer.

Jansen, K., & Trystram, D. (2016). Scheduling parallel jobs on heterogeneous platforms. *Electronic Notes in Discrete Mathematics, 55*, 9–12.

Jansen, K., & Zhang, G. (2007). Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica, 47*(3), 323–342.

Kawaguchi, T., & Kyan, S. (1986). Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing, 15*(4), 1119–1129.

Kress, D., Meiswinkel, S., & Pesch, E. (2018). Mechanism design for machine scheduling problems: Classification and literature overview. *OR Spectrum, 40*(3), 583–611.

Kumar, M., Sharma, S. C., Goel, A., & Singh, S. P. (2019). A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications, 143*, 1–33.

Liu, S. (2020). A Review for submodular optimization on machine scheduling problems. In *Complexity and approximation* (pp. 252–267). Springer.

Liu, Y., Xu, H., & Lau, W. C. (2019). 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.

Malhotra, L., Agarwal, D., Jaiswal, A., et al. (2014). Virtualization in cloud computing. *Journal of Information Technology & Software Engineering, 4*(2), 1–3.

MIP Python's Package. (n.d.). https://www.python-mip.com

Muter, İ. (2020). Exact algorithms to minimize makespan on single and parallel batch processing machines. *European Journal of Operational Research, 285*(2), 470–483.

Pinedo, M. (2012). *Scheduling* (Vol. 5). Springer.

Shin, S., Kim, Y., & Lee, S. (2015). Deadline-guaranteed scheduling algorithm with improved resource utilization for cloud computing. In *2015 12th Annual IEEE consumer communications and networking conference (CCNC)* (pp. 814–819). IEEE.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly, 3*(1–2), 59–66.

Umang, N., Bierlaire, M., & Vacca, I. (2013). Exact and heuristic methods to solve the berth allocation problem in bulk ports. *Transportation Research Part E: Logistics and Transportation Review, 54*, 14–31.

Wang, S., & Cui, W. (2021). Approximation algorithms for the min-max regret identical parallel machine scheduling problem with outsourcing and uncertain processing time. *International Journal of Production Research, 59*(15), 4579–4592.

Williamson, D. P., & Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge University Press.