Online Non-preemptive Multi-Resource Scheduling for Weighted Completion Time on Multiple Machines

Donney Fan
University of Toronto
Toronto, ON, Canada
donney.fan@mail.utoronto.ca

Ben Liang
University of Toronto
Toronto, ON, Canada
liang@ece.utoronto.ca

ABSTRACT

Jobs in computing environments have diverse and heterogeneous resource requirements. This paper presents a study of online, non-preemptive scheduling algorithms for multiple identical machines under the average weighted completion time objective. The key challenge addressed is resource allocation to jobs with non-uniform demands across multiple resource types, such as CPU, memory, and storage. We propose an online algorithm, termed Multi-Resource Interval Scheduling (MRIS) that achieves a competitive ratio of $8R(1+\epsilon)$ for the average weighted completion time, where R is the number of resource types. To the best of the authors' knowledge, this is the first theoretical competitive analysis under the considered system. We further show that the well-known priority queue algorithms can have arbitrarily bad competitive ratios in this setting. In numerical experiments using production workload traces from Microsoft Azure, the proposed algorithm is shown to significantly outperform priority queue algorithms and other state-of-the-art schedulers.

CCS CONCEPTS

• **Theory of computation → Online algorithms**.

KEYWORDS

Online algorithms, multiple resources, weighted completion time, non-preemtive scheduling

ACM Reference Format:
Donney Fan and Ben Liang. 2024. Online Non-preemptive Multi-Resource Scheduling for Weighted Completion Time on Multiple Machines. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3673038.3673149

1 INTRODUCTION

By enabling access to massive pools of computing resources, cloud computing with a large number of machines has emerged as a crucial and widely adopted paradigm in both academia and industry. However, the dynamic nature of cloud workloads, characterized by changing demands and diverse application requirements, contributes to significant challenges in the design of job schedulers in cloud computing. Long job completion times may severely impact the user experience, especially in delay-sensitive scenarios.

In addition to the difficulty of job scheduling over multiple machines, resource use patterns are often complicated by the heterogeneous multi-resource requirements of modern computing jobs. For instance, machine learning training jobs are often GPU-intensive, integer programming solvers are typically CPU-intensive, and some jobs may demand multiple resources, such as memory and storage. Another factor is that multiple jobs can be run concurrently on the same machine. Without considering multiple resources, schedulers can introduce fragmentation, which can result in poor job completion time metrics. Hence, a modern job scheduler must efficiently allocate jobs with varying resource demands to optimize the machine utilization.

When multiple jobs can be executed simultaneously on each machine, the offline problem is known to be NP-hard for both the makespan and total completion time objectives [26]. Furthermore, the packing of multiple resources is analogous to the APX-hard problem of vector bin packing, where the objective is to pack multi-dimensional vectors into the fewest number of bins, where several heuristics are known [5, 28]. At a high level, our scheduling problem is to obtain tight packing simultaneously in resources and time so that jobs can be completed quickly. While many approaches to these problems assume full knowledge of the input ahead of time, a computing scheduler should handle the *online* arrival of jobs.

In online scheduling works that do consider multiple resources, such as [15], a common theme is assuming that jobs can be preempted and migrated to other machines without penalty or delay. This allows for a simple paradigm in which the scheduler identifies the "best" jobs to process by repeatedly solving an optimization problem at each time instant. However, job preemption can incur significant costs owing to process interrupts, cache misses, context switching, and more [20]. In addition, preemption may not be viable under other system restrictions such as communication delays.

In this paper, we study *online, non-preepmtive* scheduling algorithms for multi-resource jobs over *multiple identical machines*, where a machine can process multiple jobs simultaneously subject to the machine's multi-resource capacities. This paper's main focus is the average weighted completion time objective. Our work has the following contributions:

- Our main result is a deterministic online algorithm, termed Multi-Resource Interval Scheduling (MRIS) that obtains a $8R(1+\epsilon)$ competitive ratio for the average weighted completion time (AWCT) objective, where R is the number of resource types. To the best of our knowledge, there has not been a theoretical study on this scheduling environment despite its wide appearance in practice.

ICPP '24, August 12–15, 2024, Gotland, Sweden
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673149

- We show that a popular family of priority queue algorithms in the online environment can have arbitrarily bad competitive ratios. However, we show they can be used as subroutines in MRIS to achieve bounded performance.
- Using production workload traces from Microsoft Azure, we show that our algorithms also have strong real-world performance compared with state-of-the-art algorithms.

In Section 3, we formally define the problem, and in Section 4, we show that classic priority queue algorithms can perform poorly. Section 5 presents MRIS and the proof of its competitive ratio is given in Section 6. We then investigate the real-world performance of the proposed algorithm in Section 7 and conclude in Section 8.

2 RELATED WORKS

2.1 Single Resource Online Scheduling

Significant effort has been invested in understanding scheduling problems in the *online* environment. Schwiegelshohn et al. studied the makespan minimization problem when jobs required a fixed set of processors among heterogeneous machines and showed that an adaptation to List-Scheduling [10] is 5-competitive [25]. Fox and Korupolu studied the weighted flow-time objective and used a combination of knapsack and job classification strategies to achieve a preemptive $O(1/\epsilon^2)$-competitive algorithm under $(1+\epsilon)$-speed $(1+\epsilon)$-capacity augmentation [7]. In a more practical scenario, when resources have cost functions and deadlines, Zhang et al. introduced a cost broker to exploit the discounts offered by multiple cloud service providers [31]. However, the above works considered only a single resource as a scheduling bottleneck.

2.2 Multi-Resource Online Scheduling

2.2.1 Heuristic Schedulers. There are several cloud computing schedulers that handle multiple heterogeneous resource types. Dominant Resource Fairness (DRF) satisfies several desirable fairness properties, such as sharing-incentive and envy-freeness [9]. Although it was later generalized to multiple heterogeneous servers [27], DRF does not focus on job completion time metrics. The Tetris scheduler attempts to achieve compact packing of jobs and fast job response times using a linear combination of a dot product score of the job's demands to a machine's capacity and the remaining processing time of jobs [11]. BF-EXEC reduces queuing delay by prioritizing recently released jobs and selecting machines based on the L_2-norm of the remaining resources [21]. SCheduler with Attentive Reinforcement Learning (SCARL) addressed job slowdowns through an attention mechanism [4]. Although these schedulers provide empirical studies, they do not provide any performance guarantees concerning job completion metrics.

2.2.2 Schedulers with Performance Guarantee. Several studies bound performance in multi-resource scheduling with objectives not directly related to job completion times.

In [23], the authors studied scheduling virtual machines non-preemptively using queues and assuming Poisson arrival. They used Lyapunov analysis and showed that their algorithm achieved a guaranteed fraction of the maximum throughput on each machine. The authors also studied a randomized algorithm for the same problem that has lower complexity using Poisson clocks [24]. Zheng and Shroff investigated multi-resource jobs with deadlines for a computing cloud [32]. Under a model in which the partial and preemptive execution of tasks yields partial profit, the authors provided a 2-competitive algorithm with respect to the profit achieved by solving a convex optimization problem in each time slot. Yin et al. leveraged resource augmentation to provide a competitive ratio in a profit maximization problem for edge-clouds systems where jobs yield a profit if completed before a deadline [30]. Our work differs from these works in that it considers job completion times in our objective, and it further differs from [23, 24] by avoiding the Poisson arrival assumption, [32] by disallowing preemption, and [30] by not directly using resource augmentation.

Online algorithms that directly optimize for some form of job completion time metrics are presented in [6, 15, 16, 29]. Im et al. studied a general problem in [16] where each machine is assigned a polytope constraint. Their approach used the proportional fairness algorithm to solve a resource allocation problem at each time instance and assigned processing rates to jobs, which achieved an $O(1)$ competitive ratio for total weighted completion time minimization, but required preemption. In the same work, they used resource augmentation and showed that for a particular problem instance, for any $\epsilon \in (0, 1/2)$, it was $O(1/\epsilon^2)$-competitive for minimizing the weighted flow time under $(e+\epsilon)$-speed augmentation. This was also extended to the case of multiple clusters in [15]. Xu et al. combined Tetris and cloning of tasks to produce an $O(1/\epsilon)$-competitive algorithm for the flow time under $(2+\epsilon)$-capacity augmentation for a single machine [29]. When jobs have unit processing times and the system contains a single machine, Epstein et al. provided online algorithms with competitive ratios that are sublinear in the number of resource types when minimizing the makespan [6]. Although these works show that resource augmentation can be used to obtain bounded ratios, it is unclear how researchers or algorithm designers should interpret the degree of augmentation used [3]. Our work differs from [16] by disallowing preemption and from works [15, 29] by not directly using resource augmentation, and also from [6, 29] by allowing multiple machines.

3 PROBLEM DEFINITION

We are interested in an environment where N independent jobs arrive over time to be scheduled on M identical machines. Each job j is associated with a processing time p_j, a weight w_j, a release time r_j, and a resource demand d_{jl} for the l-th resource. Jobs arrive over time in an online fashion and their parameters are not revealed to the scheduler until time r_j. Each machine has R orthogonal resources and capacity U_l for resource l. Machines can process multiple jobs simultaneously if the jobs being processed do not exceed the resource capacity. We assume that resources are infinitely divisible. Without loss of generality, we scale p_j by the minimum processing time and normalize the machine capacity for each resource to one. Then, we have $p_j \geq 1$, $d_{jl} \leq 1$, and $U_l = 1$.

As explained in Section 1, we believe it is important to study non-preemptive scheduling to avoid costs, such as context switching or migration, so we require that the jobs cannot be stopped once they start being processed. We allow the scheduler to collect jobs and defer scheduling decisions at a later time. Each job must be assigned a machine and a start time $S_j \geq r_j$. Its completion time C_j

satisfies $C_j = S_j + p_j$. A schedule (i.e., an assignment of start times to jobs) satisfies resource constraints when the machine resource capacity is not exceeded at any point in time. That is, for every machine, $\sum_{j:S_j \le t < C_j} d_{jl} \le 1$ for all l and all points in time t.

In this problem setting, our objective is to minimize $\frac{1}{N}\sum_j w_j C_j$, the average weighted completion time (AWCT). We measure the performance of an online algorithm using the common *competitive ratio.* Let I be a problem instance (i.e., a collection of unscheduled jobs with normalized demands), $\mathrm{ALG}(I)$ be the value of the objective achieved by an algorithm scheduling I, and $\mathrm{OPT}(I)$ be the value achieved by an optimal scheduler that has full knowledge of all job parameters at time zero. An online algorithm to a minimization problem is said to be ρ-competitive, if for all problem instances I we have $\mathrm{ALG}(I) \le \rho\, \mathrm{OPT}(I)$. We also use an asterisk (*) to indicate the values chosen by an optimal scheduler.

4 NON-COMPETITIVENESS OF PRIORITY-QUEUE ALGORITHMS

As illustrated in Section 2, many algorithms have been proposed to tackle online scheduling, and several of these fall under a class of priority queue algorithms, such as Smallest-Volume-First (SVF) [17] and Shortest-Job-First (SJF). Notably, the popular Tetris scheduler, when not given the power of preemption, also falls under this class [11].

Such algorithms maintain a queue of jobs that have arrived and schedule them in the following manner: Let $\mathcal{T}$ be the set of time points containing an "event", i.e., where a job arrives or completes on some machine. On every event $t \in \mathcal{T}$, construct the queue $Q_t = \{j \mid r_j \le t, j \text{ not yet scheduled}\}$, and order the jobs in the queue according to some heuristic, such as sorting by non-decreasing order of job processing times in SJF. Starting from the head of the queue, scan and repeatedly schedule jobs if they are feasible at time t. We call this class of algorithms Priority-Queue (PQ).

Although it is possible for these algorithms to have good average or typical performance, we show that there always exists an adversarial input instance of jobs, such that the competitive ratio of this class of algorithms is arbitrarily bad.

Lemma 4.1. *The competitive ratio of the Priority-Queue class of algorithms for minimizing the AWCT is $\Omega(N)$.*

Proof. Let there be N jobs and one machine. The first job is released at time zero and has demand one for each resource and a processing time $p \ge 1$. The remaining $N-1$ jobs are released at time $\varepsilon > 0$ and have demand $\frac{1}{N-1}$ for each resource and processing time 1. We also set $w_j = 1$ for all jobs.

At time $t = 0$, since there is an event, PQ schedules the first job. The other $N-1$ jobs must wait until the first job is completed and these jobs can be processed simultaneously. This gives a total completion time of

$$\sum_j w_j C_j = p + (N-1)(p+1).$$

On the other hand, the optimal offline scheduler would skip the first job until the end. Letting C_j^* be the optimal completion time of jobs, it yields a total weighted completion completion time of

$$\sum_j w_j C_j^* = (N-1)(1+\varepsilon) + 1 + \varepsilon + p.$$

The competitive ratio is then lower bounded by

$$\frac{\sum_j w_j C_j}{\sum_j w_j C_j^*} = \frac{p + (N-1)(p+1)}{(N-1)(1+\varepsilon) + 1 + \varepsilon + p} \in \Omega\left(\frac{Np}{N+p}\right).$$

Letting $p = N$ we have the result, and we note the result is independent of the number of resources. □

5 MULTI-RESOURCE INTERVAL SCHEDULING

The weakness of the PQ approach is its lack of flexibility and adaptability. By constantly processing a job whenever resources are available, it operates under the assumption that the current job is the most appropriate at that moment. However, in dynamic environments, where priorities may shift, more critical tasks can emerge, and this rigid approach can lead to highly suboptimal scheduling decisions. Without the ability to preemptively interrupt ongoing tasks to accommodate higher-priority jobs, PQ risks committing prematurely to poor schedules.

Introducing a delay before making scheduling decisions allows the scheduler to gather more information about the pending tasks or the system's state, allowing it to potentially make more informed decisions and avoid situations in which a better-suited job arrives shortly after committing to a less favorable one, as exemplified in Lemma 4.1. By exercising patience and waiting for additional context, the scheduler can more effectively schedule jobs that reduce the weighted completion time of jobs. However, incurring a delay increases the completion time of jobs; therefore, a judicious design is necessary.

In [13], Hall et al. presented a framework for online scheduling that executes over multiple iterations, each using a geometrically increasing time period. In each period, it solves a subproblem that identifies a maximum-weighted subset of jobs that can be scheduled within a certain deadline. The scheduler then schedules that subset of jobs in an offline manner while simultaneously waiting for jobs to arrive and be considered in the next iteration. This process repeats until completion.

In this work, we adopt the general framework of [13] to avoid the rigidity of PQ. However, it remains unclear how to (i) identify the subset of jobs, (ii) find a guaranteed deadline for scheduling such subset in each interval, (iii) maintain feasibility, and (iv) do so in fully polynomial time. Furthermore, [13] suggested scheduling each subset of jobs in their own disjoint intervals. This can result in poor resource use, as in a typical case, the collected subset may not require the use of the entire interval. Instead, we are interested in scheduling such jobs as early as possible (backfilling).

To solve these challenges, the proposed Multi-Resource Interval Scheduling (MRIS) algorithm uses two subroutines. In each iteration, we first approximate a variant of the knapsack problem to identify a maximum-weighted subset of jobs. Then, we use a makespan algorithm to schedule the subset before a deadline set by the knapsack capacity. The details of these two subroutines are presented in Sections 5.1 and 5.2, respectively, followed by the MRIS algorithm description in Section 5.3.

5.1 Knapsack Constraint Approximation

Maximizing the weight of a subset of jobs is akin to solving a knapsack problem, where item profits are the weights. To be competitive with the optimal scheduler, we seek solutions that can exactly identify the subset of jobs with the maximum weight.

It is well known that the knapsack problem can be optimally solved via dynamic programming with best-known running time $O(\zeta n)$ for knapsack capacity ζ and n number of items [22]. Unfortunately, its runtime is pseudo-polynomial. To design a fully polynomial time solution, one can design approximation algorithms, which are guaranteed to achieve a fraction of the optimal weight.

To design MRIS, we are more interested in knapsack *constraint approximation*, first suggested by Lawler [19] and later expanded to the general *resource augmentation* model [18]. In these models, the designed algorithms are given additional capacity compared with the optimal. Within the context of our algorithm, we use constraint approximation to obtain a subset of jobs with matching or greater total weight compared with the optimal knapsack solution, so that in each iteration, we can match at least the total weight scheduled by the optimal scheduler.

However, in our design, we must avoid direct use of resource augmentation, as in practice, it is meaningless for our algorithm to have "additional knapsack capacity" compared with the optimal scheduler. Towards this end, we define the volume of a job. Denote the total demand of a job j by $u_j = \sum_{l=1}^{R} d_{jl} \leq R$. We define the volume of job j as $v_j = p_j u_j$ and the total volume of jobs of an instance I as $V_I = \sum_{j \in I} v_j$. Then, we equate the size of each job with its volume, and the knapsack capacity will be some volume capacity that is adaptively updated in each iteration. In this way, the use of resource augmentation would imply using additional "volume", which we will later show in Section 6.2 can be used to bound the makespan of jobs scheduled.

More formally, in each iteration k, given a set of jobs $\mathcal{J}_k$ and a knapsack capacity ζ_k that is adaptively computed within MRIS, we find $\mathcal{B}_k = \{j \mid x_j = 1\}$ where x_j solves the following problem:

$$\textbf{P1:} \quad \max_{x_j \in \{0,1\}} \sum_{j \in \mathcal{J}_k} w_j x_j \quad \text{subject to} \sum_{j \in \mathcal{J}_k} v_j x_j \leq \zeta_k$$

We note that we can use any knapsack constraint approximation algorithm here.

We now present Constraint-Approximate Dynamic Programming (CADP), which modifies Ibarra & Kim's fully polynomial time approximation scheme [14]. For some small error parameter $\epsilon > 0$, and knapsack capacity ζ, set $K = \zeta n/\epsilon$. For each item j define a downscaled and rounded-down size $\hat{v}_j = \lfloor v_j / K \rfloor$. We run dynamic programming using $\{\hat{v}_1, \ldots, \hat{v}_n\}$ as our size inputs with a knapsack of capacity $\hat{\zeta} = \lfloor \zeta / K \rfloor$ and obtain an exact solution. We then output the unscaled items that correspond to the solution. We will show in Section 6.1 that CADP incurs $1 + \epsilon$ times the knapsack capacity, which adds only slightly to the AWCT of MRIS.

In Section 6.1, we consider an alternative greedy heuristic that requires augmentation at twice the capacity. We ultimately select dynamic programming because it yields a better competitive ratio.

Algorithm 1 MRIS Algorithm

```
k ← 0
while all jobs are not yet scheduled do
    $\mathcal{J}_k = \{j \mid p_j \leq \gamma_k, r_j \leq \gamma_k, j$ not yet scheduled$\}$
    $\zeta_k \leftarrow RM\gamma_k$
    Find $\mathcal{B}_k \subseteq \mathcal{J}_k$ by solving P1 with knapsack capacity $\zeta_k$ using
        CADP.
    Schedule $\mathcal{B}_k$ using Priority-Queue starting at time $\gamma_k$.
    $k \leftarrow k + 1$
end while
```

5.2 Priority-Queue for Makespan

Once we solve **P1**, we require a makespan algorithm to schedule the jobs while simultaneously providing an upper bound on the length of the schedule based on the volume of jobs.

The well-known List-Scheduling algorithm was first proposed by Graham [10]. In an environment without any resource constraints, where each machine processes one job at a time, it is $(2 - 1/M)$-approximate. List-Scheduling was later extended to an environment with R auxiliary resources, where it was shown to be $(R + 1)$-approximate when $M \geq N$ (i.e., the number of machines is not a constraint so that all jobs can be processed in parallel subject to the resource constraint) [8]. One can interpret their model of $M \geq N$ machines being allowed to process only one job as one large single machine being associated with R resources. We borrow their scheduling ideas while extending to multiple machines and incorporating the volume heuristic so it can be used with knapsack.

In fact, the PQ family of algorithms presented in Section 4 encompasses an extension of List-Scheduling to our resource constraints on multiple machines. We use PQ to schedule the jobs found by the knapsack constraint approximation algorithm, with the only modification being that the release times of these jobs are zero to PQ, effectively scheduling those jobs offline. Whenever a machine has available resource capacity, either at the start of a schedule or when a job is completed, we scan the list of jobs that can be scheduled feasibly at that time. If there is such a job, schedule it to start on that machine, remove it from the list, and recurse until the list is scanned. PQ used this way has runtime complexity $O(N^2)$.

We remark here that despite PQ having poor performance when used standalone (see Section 4), when we combine it with our modified knapsack problem in MRIS, with a controlled set of intervals where we execute these subroutines, we can achieve a bounded competitive ratio to be shown in Section 6.

5.3 MRIS Algorithm Description

We are now ready to present the proposed MRIS algorithm. It runs for multiple iterations, indexed by k, until completion. Let γ_k be a geometric series that satisfies $\gamma_{k+1} - \gamma_k \geq \gamma_k$ and $0 < \epsilon < 1$ be an error parameter that parameterizes the solution to **P1**.

For the k-th iteration, we consider the set of jobs $\mathcal{J}_k$ that have arrived by time γ_k but have yet to be scheduled. Set the knapsack capacity ζ_k to $RM\gamma_k$, the maximum volume of jobs we can schedule with a bounded makespan. We identify a subset of jobs $\mathcal{B}_k \subseteq \mathcal{J}_k$ with maximum weight subject to the knapsack capacity ζ_k and schedule them using PQ while using backfilling to find the earliest possible times a job can be placed in the schedule. As mentioned earlier, this allows the start times of jobs in the period of one iteration to enter periods of previous iterations. The above steps of MRIS are summarized in Algorithm 1.

Next, we investigate the runtime complexity of MRIS. For each k, as will be shown in Lemma 6.1, solving knapsack using a CADP has runtime complexity $O(|\mathcal{J}_k|^2/\epsilon)$, and PQ has runtime $O(|\mathcal{J}_k|^2)$. MRIS solves $O(N)$ such problems. Therefore, the total runtime complexity is $O(N^3/\epsilon)$. Furthermore, with some loss in the competitive ratio, one can use a greedy solution (see Section 6.1) to the knapsack problem and obtain $O(N^2 \log N)$ runtime.

6 COMPETITIVE ANALYSIS

Our competitive analysis of MRIS echoes the presentation of its design in Section 5. We first derive a bound on CADP, then provide upper bounds on the makespan of PQ, and finally combine these results with the geometric interval sequence to show that MRIS is $8R(1+\epsilon)$-competitive.

6.1 Knapsack Constraint Approximation

In the original knapsack problem, we are given an instance I containing n items, each item j defined by its size v_j and weight w_j. We wish to identify $\mathcal{J} \subseteq I$ that maximize the total weight of the knapsack subject to its capacity ζ. As mentioned in Section 5.1, we seek a fully polynomial time constraint approximation, where we are allowed knapsack capacity larger than ζ.

Lemma 6.1. *CADP obtains the optimal knapsack weight using knapsack capacity $(1+\epsilon)\zeta$ and runs in fully polynomial time.*

Proof. Let $\mathcal{J}$ be the items found by CADP. Since the weights has not changed and dynamic programming is exact, $\sum_{j\in\mathcal{J}} w_j$ is optimal. For each item, we scaled down its size by a factor of K and rounded down, so any item satisfies $K\hat{v}_j \le v_j$ with the difference at most K. Therefore, the total size of items returned by the algorithm is at most nK. Then, we can bound the required capacity:

$$\begin{aligned}\sum_{j\in\mathcal{J}} v_j &\le nK + K\sum_{j\in\mathcal{J}} \hat{v}_j \\ &= nK + K\sum_{j\in\mathcal{J}} \lfloor v_j/K \rfloor \\ &\le nK + \sum_{j\in\mathcal{J}} v_j \\ &= \epsilon\zeta + \sum_{j\in\mathcal{J}} v_j \\ &\le (1+\epsilon)\zeta\end{aligned}$$

The runtime is $O(n\lfloor \zeta/K \rfloor)$ or $O(n^2/\epsilon)$. □

Remark 1. One might be interested in algorithms that have lower runtime complexity. We show that the classic $O(n\log n)$ greedy algorithm can reach the optimal knapsack weight by allowing twice the knapsack capacity. This algorithm sorts the items using the ratio w_j/v_j and selects items based on the non-increasing order of this ratio up to the k-th item that cannot be added to the knapsack. The output selects the better of the items $\{1,\ldots,k-1\}$ or $\{k\}$. If a fraction of the k-th item was permitted, the knapsack output could match or exceed the optimal knapsack weight. Thus, by allowing a knapsack with capacity 2ζ, we can obtain the optimal weight. However, since the greedy algorithm requires a larger capacity, it yields an inferior competitive ratio compared with CADP (see also Section 7.4). Therefore, we focus instead on CADP.

6.2 Priority-Queue for Makespan Scheduling

As explained in Section 5.2, PQ is used in MRIS to solve an offline problem on a subset of jobs, where the goal is to minimize the makespan for each iteration k. We first present lower bounds on the makespan, which can be viewed as a generalized case of lower bounds of the multiple strip packing problem [2].

Lemma 6.2. *Let $C^*_{\max} = \max_j\{C^*_j\}$ be the optimal makespan for scheduling an offline problem instance I. Then $C^*_{\max} \ge \frac{V_I}{RM}$.*

Proof. Let $\mathcal{J}_i$ be the jobs scheduled on machine i by an optimal scheduler. We compute the total volume of jobs:

$$\begin{aligned}V_I &= \sum_{i=1}^{M}\sum_{j\in\mathcal{J}_i} p_j \sum_{l=1}^{R} d_{jl} \\ &= \sum_{l=1}^{R}\sum_{i=1}^{M}\sum_{j\in\mathcal{J}_i} p_j d_{jl} \\ &\le \sum_{l=1}^{R}\sum_{i=1}^{M} 1\cdot C^*_{\max} \\ &= RMC^*_{\max}.\end{aligned}$$

The inequality is given by observing $\sum_{j\in\mathcal{J}_i} p_j d_{jl}$ as the contribution of resource l to the total volume of jobs. One can view this problem as similar to the strip-packing problem, where each job in $\mathcal{J}_i$ is modelled as a rectangle with height d_{jl} and width p_j. This problem aims to find the smallest packing in a rectangle of height one and unlimited width. The inequality states that each rectangle's total area (or volume) must be contained within the area of the optimal strip, which has dimension $1\cdot C^*_{\max}$. □

Lemma 6.3. *Let $C_{\max}$ be the makespan achieved by PQ, $C^*_{\max}$ be the makespan achieved by the optimal scheduler, and $p_{max} = \max_{j\in I}\{p_j\}$. Then $C_{\max} \le \max\{2p_{\max}, 2V_I/M\} \le 2RC^*_{\max}$, where V_I is the volume of jobs in the scheduled instance.*

Proof. We study two cases. First assume that $C_{\max} \le 2p_{\max}$. Then, the result holds trivially as $2p_{\max} \le 2C^*_{\max} \le 2RC^*_{\max}$.

In the second case, we assume $C_{\max} > 2p_{\max}$. Let $\mathcal{J}_{it}$ be the set of jobs active on a machine i at time t and Let $U_{ilt} = \sum_{j\in\mathcal{J}_{it}} d_{jl}$ be the total demand of all active jobs at a machine i on resource l at time t. Let $t \in [0, C_{\max}/2]$ and $\tau = t + C_{\max}/2$. By our assumption $p_{\max} < C_{\max}/2$, so any job in $\mathcal{J}_t$ cannot belong to $\mathcal{J}_\tau$. There exists some l on every machine i such that $U_{ilt} + U_{il\tau} > 1$. Otherwise, if $U_{ilt} + U_{il\tau} \le 1$ for every resource l, an active job at time τ could have been scheduled at time t by PQ, since there would not have been a violation of the resource constraint. We compute the total volume of jobs scheduled by PQ:

$$V_I = \int_0^{C_{\max}} \sum_{i=1}^{M}\sum_{j\in\mathcal{J}_{it}} u_j \, dt$$

$$
\begin{aligned}
&= \int_0^{C_{\max}} \sum_{i=1}^{M} \sum_{l=1}^{R} U_{ilt}\, dt \\
&= \int_0^{C_{\max}/2} \sum_{i=1}^{M} \sum_{l=1}^{R} \left(U_{ilt} + U_{il\tau}\right)\, dt \\
&> \int_0^{C_{\max}/2} \sum_{i=1}^{M} 1\, dt \\
&= \frac{C_{\max} M}{2}
\end{aligned}
$$

Therefore by Lemma 6.2 we have $C_{\max} < 2V_I/M \leq 2RC^*_{\max}$. □

The following lemma shows that the upper bound on the total volume of jobs scheduled by PQ cannot be improved.

Lemma 6.4. *Let $C_{\max}$ be the makespan achieved by* Priority-Queue *and V_I the volume of jobs of some scheduled instance. There exists I where $C_{\max} \leq \max\{2p_{\max}, 2V_I/M\}$ is tight.*

Proof. Let there be a single machine. As a first trivial case, let I contain 1 job with resource demand of $1/2$ in the first resource and 0 in the remaining $R - 1$ resources and a processing time of p. The makespan achieved by PQ is p and $2V_I/M = p$. For a nontrivial case, let I contain N jobs with $p_j = p$ for all jobs, and each job has demand $(1/2 + \delta)$ for some $\delta > 0$ in the first resource and no demand for the remaining $R - 1$ resources. The machine can process no more than one job at once, so the makespan achieved is Np. On the other hand, the total volume of jobs is $Np(1/2 + \delta)$, so we have $2V/1 = Np(1 + 2\delta)$, which approaches the bound as $\delta \to 0$. □

6.3 Competitive Ratio of MRIS

We derive the competitive ratio of MRIS by showing that, for each iteration k, we can find a subset of jobs of maximal weight that was completed by the optimal scheduler in one period of time and schedule that subset using a slightly longer period of time.

Lemma 6.5. *Let $\mathcal{J}_k$ be the set of arrived jobs found by MRIS on line 3 of Algorithm 1, and suppose there exists a set of jobs $\mathcal{B}^*_k \subseteq \mathcal{J}_k$ of total weight W that can be completed by an optimal scheduler within γ_k time steps. Then, for any constant $0 < \epsilon < 1$, MRIS schedules a set of jobs $\mathcal{B}_k \subseteq \mathcal{J}_k$ of weight at least W within $2R(1 + \epsilon)\gamma_k$ time steps.*

Proof. Let $\mathcal{B}^*_k$ be the set of jobs with weight W be scheduled feasibly by time γ_k. To construct our desired schedule, we need to find a subset of jobs with weight at least W and makespan bounded by γ_k, which is the knapsack problem discussed before. Its optimal solution gives a set of jobs $\mathcal{B}_k$ of maximal weight with total volume $V_{\mathcal{B}_k} \leq RM\gamma_k$ and by Lemma 6.2 is a lower bound on the optimal makespan for scheduling these jobs, which is γ_k. That is, by solving the knapsack, we can identify a set of jobs with at least as much weight as the optimal would schedule on the machines by time γ_k.

In Lemma 6.1, we show that CADP can achieve at least as much as the optimal knapsack weight, given $(1 + \epsilon)$ slack on the volume constraint. Finally, we use PQ, which schedules $\mathcal{B}_k$ by time $\max\{2p_{\max}, 2V_{\mathcal{B}_k}/M\}$ where $p_{\max} = \max_{j \in \mathcal{B}_k}\{p_j\}$. Since $p_{\max} \leq \gamma_k$ and from CADP we have $V_{\mathcal{B}_k}/RM \leq (1 + \epsilon)\gamma_k$, we can feasibly schedule the jobs by time $2R(1 + \epsilon)\gamma_k$. □

The following lemma uses Lemma 6.5 repeatedly to show that the total weight of jobs that we schedule over time is at least as large as the optimal at specific time points.

Lemma 6.6. *Let $\mathcal{B}_k$ be the set of jobs computed at time γ_k under MRIS and let $\mathcal{B}^*_k$ be the set of jobs in the optimal offline solution which are completed in the interval $[\gamma_{k-1}, \gamma_k)$. Then for any $i \leq k$, we have*

$$\sum_{i=1}^{k} \sum_{j \in \mathcal{B}_i} w_j \geq \sum_{i=1}^{k} \sum_{j \in \mathcal{B}^*_i} w_j. \tag{1}$$

Proof. Since $p_j \geq 1$, no jobs can be completed before time γ_0. We consider each interval $[\gamma_{k-1}, \gamma_k)$. Let $\mathcal{J}_k$ be the set of pending jobs released by time γ_k, but have not been scheduled. MRIS constructs $\mathcal{B}_k \subseteq \mathcal{J}_k$ at γ_k.

Let $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{B}^*_i \setminus \bigcup_{i=1}^{k-1} \mathcal{B}_i$ be the set of jobs the optimal scheduler has completed by time γ_k, but our algorithm has not scheduled in the k-th iteration. By definition, for each job in $\mathcal{S}$, it also belongs to $\mathcal{J}_k$ and therefore $\mathcal{S} \subseteq \mathcal{J}_k$. By Lemma 6.5 CADP constructs $\mathcal{B}_k \subseteq \mathcal{J}_k$ with weight at least as large as the total weight of jobs in $\mathcal{S}$. Lemma 6.5 is applicable because, by definition, $\mathcal{S}$ can be completed by the optimal scheduler within γ_k time steps. Therefore, the total weight of jobs in $\bigcup_{i=1}^{k} \mathcal{B}_i$ found by MRIS is at least the total weight of jobs in $\bigcup_{i=1}^{k} \mathcal{B}^*_i$.

It is possible that some jobs in $\bigcup_{i=1}^{k} B^*_i$ have been scheduled by our algorithm in previous phases, and by definition, these jobs would have been released by time γ_k. Ignoring those jobs in the construction of $\mathcal{B}_k$, as in the definition of $\mathcal{J}_k$, would not increase the length of the schedule. □

We will also need the following elementary result to help arrive at our theorem.

Lemma 6.7. *Let $\{x_k\}_{k=1}^{K}$ and $\{y_k\}_{k=1}^{K}$ be finite non-negative sequences that satisfy*

$$\sum_{k=1}^{K} x_k = \sum_{k=1}^{K} y_k, \tag{2}$$

and for any $j = 1, \ldots, K$ they also satisfy

$$\sum_{k=1}^{j} x_k \geq \sum_{k=1}^{j} y_k. \tag{3}$$

Then for any non-decreasing non-negative sequence $\{z_k\}_{k=1}^{K}$ we have

$$\sum_{k=1}^{K} z_k x_k \leq \sum_{k=1}^{K} z_k y_k. \tag{4}$$

Proof. We show the result by induction.
Base Case $K = 1$: The result trivially holds: $z_1 x_1 \leq z_1 y_1$.
Inductive Step: Let $\{x_k\}$ and $\{y_k\}$ be sequences of length $K+1$ that satisfy the assumptions (2) and (3) and $\{z_k\}$ be a non-decreasing non-negative sequence of length $K + 1$.

Since $\sum_{k=1}^{K} x_k \geq \sum_{k=1}^{K} y_k$ and $\sum_{k=1}^{K+1} x_k = \sum_{k=1}^{K+1} y_k$ we have $x_{K+1} \leq y_{K+1}$. Multiply through by z_{K+1} to obtain

$$x_{K+1} z_{K+1} \leq y_{K+1} z_{K+1}. \tag{5}$$

We re-write the sum by adding and subtracting terms:

$$\begin{aligned}\sum_{k=1}^{K+1} z_k x_k &= (z_1 - z_2)x_1 + (z_2 - z_3)(x_1 + x_2)\\ &\quad + (z_3 - z_4)(x_1 + x_2 + x_3)\\ &\quad + \cdots\\ &\quad + (z_K - z_{K+1})(x_1 + x_2 + \cdots + x_K)\\ &\quad + z_{K+1}(x_1 + x_2 + \cdots + x_{K+1})\\ &= \sum_{k=1}^{K}\sum_{j=1}^{k}(z_k - z_{k+1})x_j + z_{K+1}\sum_{j=1}^{K+1} x_j\\ &\le \sum_{k=1}^{K}\sum_{j=1}^{k}(z_k - z_{k+1})y_j + z_{K+1}\sum_{j=1}^{K+1} y_j\\ &= \sum_{k=1}^{K+1} z_k y_k\end{aligned}$$

We first observe that if $a \ge b$ and $c \le 0$, then $ac \le bc$. From the non-decreasing property, we know that $z_k - z_{k+1} \ge 0$ and from non-negativity $z_K \ge 0$. Therefore, by applying (3) to the first sum and (5) with the inductive hypothesis to the second sum, we have the inequality and thus the result. □

We are now ready to present the main result of this paper.

Theorem 6.8. *MRIS is an $8R(1+\epsilon)$-competitive algorithm for the weighted average completion time objective.*

Proof. Let $\sum_j w_j C_j$ be the sum of completion times achieved by MRIS. Let $\mathcal{B}_k$ be the set of jobs scheduled by the algorithm at iteration k. MRIS schedules $\mathcal{B}_k$ between time γ_k and $\gamma_k + T_k$, where T_k is the time period of the schedule that PQ schedules jobs in the k-th iteration. As we use backfilling, our use of PQ does not follow Lemma 6.2. However, as backfilling cannot increase the makespan of PQ, in the worst case it provides no benefit to MRIS, resulting in no overlap between jobs scheduled in one iteration to the next. In this worst case, $T_k = 2R(1+\epsilon)\gamma_k$ is achieved exactly. Let t_k be the makespan of the total schedule during the k-th iteration. Since γ_k satisfies $\gamma_{k+1} - \gamma_k \ge \gamma_k$, before scheduling $\mathcal{B}_k$, the makespan is

$$t_{k-1} \le \sum_{\kappa=0}^{k-1} 2R(1+\epsilon)\gamma_\kappa \le 2R(1+\epsilon)\gamma_k.$$

Therefore processing $\mathcal{B}_k$ finishes before time $t_k \le 4R(1+\epsilon)\gamma_k$. Let $\gamma_k = \alpha^k$ and suppose the algorithm makes a total of $K+1$ iterations (i.e., an optimal scheduler scheduled all jobs by time γ_K). For notational convenience let $\gamma_{-1} = 0$.

We sum over all the iterations:

$$\begin{aligned}\sum_{j=1}^{N} w_j C_j &\le \sum_{k=0}^{K} 4R(1+\epsilon)\gamma_k \sum_{j\in\mathcal{B}_k} w_j\\ &\le 4R(1+\epsilon)\alpha \sum_{k=0}^{K}\gamma_{k-1}\sum_{j\in\mathcal{B}_k} w_j\\ &\le 4R(1+\epsilon)\alpha \sum_{k=0}^{K}\gamma_{k-1}\sum_{j\in\mathcal{B}_k^*} w_j\end{aligned}$$

The third step is an application of Lemma 6.6 and Lemma 6.7 where $x_k = \sum_{j\in\mathcal{B}_k} w_j$, $y_k = \sum_{j\in B_k^*} w_j$ and $z_k = \gamma_{k-1}$. Let C_j^* be the optimal offline solution's completion time of job j. For each job let $[\gamma_{k_j-1}, \gamma_{k_j})$ be the interval that contains C_j^*. Then we have

$$\sum_{k=0}^{K}\gamma_{k-1}\sum_{j\in\mathcal{B}_k^*} w_j = \sum_{j=1}^{N}\gamma_{k_j-1}w_j \le \sum_{j=1}^{N} w_j C_j^*.$$

The equality is a result of a difference in accounting. The first double sum adds the weight of jobs the optimal algorithm schedules in each interval and assumes it completes by time γ_{k-1}. The second sum uses γ_{k_j} to index a single sum over j, maintaining the assumption the a job that finishes in $[\gamma_{k_j-1}, \gamma_{k_j})$ by the optimal algorithm is lower bounded by γ_{k_j-1}. These steps imply $\sum_j w_j C_j \le 4R(1+\epsilon)\alpha\sum_j w_j C_j^*$. We select $\alpha = 2$ as the smallest such base that satisfies $\gamma_{k+1} - \gamma_k \ge \gamma_k$ and therefore MRIS is $8R(1+\epsilon)$-competitive. □

6.4 Discussion

Remark 2. A curious reader might be interested in why we used PQ instead of the $(R+1)$-approximation result from Garey and Graham [8] for makespan scheduling. This is because one cannot use a makespan approximation ratio to prove Lemma 6.5, since after constructing $\mathcal{B}_k$ from $\mathcal{J}_k$ from the knapsack problem, we do not use the optimal makespan of $\mathcal{B}_k$ to conclude PQ can schedule the jobs by some multiple of γ_k.

Remark 3. One might be interested in whether the performance bound can be improved in the special case where $p_j = 1$ for all jobs. In this case, we can use bin-packing algorithms that are known to have sublinear asymptotic approximation ratios with respect to the number of resources [1]. However, for the reasons mentioned above, we cannot directly use the asymptotic ratio or obtain tighter upper bounds of a form similar to Lemma 6.3.

Remark 4. There may be cases in which the makespan of jobs may also be an important consideration. Although our main problem is the average weighted completion time objective, our algorithm is also bounded with respect to the makespan.

Lemma 6.9. *MRIS is $8R(1+\epsilon)$-competitive for minimizing makespan.*

Proof. Similar to the proof of Theorem 6.8, let γ_K be the time at which the optimal makespan lies in the range $[\gamma_{K-1}, \gamma_K)$. MRIS generates a feasible schedule, and the last job completes by time $4R(1+\epsilon)\gamma_K$. Setting $\gamma_k = 2^k$, the competitive ratio is $8R(1+\epsilon)$. □

In a sense, our algorithm *simultaneously* optimizes the average weighted completion time and makespan objectives.

7 TRACE-DRIVEN EVALUATION

While Section 6 provides a worst-case performance guarantee, in this section, we evaluate the real-world behavior of MRIS with trace-based simulation. We observe that it exhibits strong performance compared with state-of-the-art cluster schedulers.

7.1 Experimental Setup

We use the Microsoft Azure traces for virtual machine (VM) allocation designed to evaluate packing algorithms [12]. It contains VM requests over 14 days with start and completion times and integer priorities that we interpret as a weight. There is also a table of possible VM types for each request, with varying degrees of resource demands. The dataset includes five resource types: CPU, memory, hard drive (HDD) space, solid-state drive (SSD) space, and network bandwidth. The dataset maps a VM type to resource usage as a fraction of total capacity for up to 34 different machine types. VM requests do not include p_j, so we determine it by subtracting the start time from the completion time. There is a wide distribution of p_j, ranging from a few seconds to 90 days.

Several modifications to the dataset are required. Due to the lack of a single machine type encompassing all VM types, we randomly sample a machine type for each VM type, preserving the original data. We use the first 4.096 million jobs of the dataset, such that the last job is released at nearly 12.5 days. The system can evolve beyond this time, until all jobs are scheduled and completed. The dataset does not contain information regarding the number of machines used to schedule these jobs; therefore, we set $M = 20$ as the default. Lastly, we ignore jobs with negative start times.

To keep simulation time manageable, we choose a factor f to "downsample" the job submissions to simulate a smaller number of jobs arriving over the 12.5 days. For example, to simulate 64000 jobs arriving over the 12.5-day time window, we sort the jobs by r_j and select every $f = 64$ jobs. To strengthen the statistical significance of our results, we obtain multiple sampled job sets by ignoring the first Δ arrival of jobs during downsampling, where $\Delta \in \mathbb{Z}$ is drawn uniformly from the interval $[0, f)$ without replacement. For every data point we obtain 10 sampled job sets, plot the mean, and shade the 95% confidence interval in the following figures. Lastly, jobs requiring SSD space do not require HDD space and vice versa, so we combined these into one storage resource demand.

7.2 Comparison Benchmarks

For benchmarks, we select the following algorithms:

- Priority-Queue (PQ)
- Tetris [11]
- BF-EXEC [21]
- Collect-All-Priority-Queue (CA-PQ)

As mentioned before, Tetris is akin to PQ. At each time instance, when resources are available on a machine i, Tetris computes an alignment score for each job j not yet scheduled as $a_j + \varepsilon v_j$ where a_j is the dot product score of the remaining resources of machine i and v_j. It then recursively assigns jobs until no more jobs can be scheduled. We note that Tetris uses the notion of the *remaining* volume because of their preemption assumption, whereas we adapt their algorithm to our environment. In effect, jobs are sorted by SVF, selected by the alignment scores, and cannot be preempted.

The BF-EXEC scheduler gives preference jobs that have recently arrived and uses SJF when selecting jobs from the queue. Upon arrival of a new job, the scheduler assigns the job to the machine with the lowest L_2-norm of remaining resources, if feasible. Upon departure of a job, the scheduler recursively finds the shortest job in the queue and assigns it to the machine of the recently departed job, given sufficient resource capacity.

As MRIS exercises patience in scheduling jobs, one may be interested in the extreme case in which we wait for all jobs to arrive before scheduling all jobs together. CA-PQ fills this role and is given additional knowledge of the last job's release time.

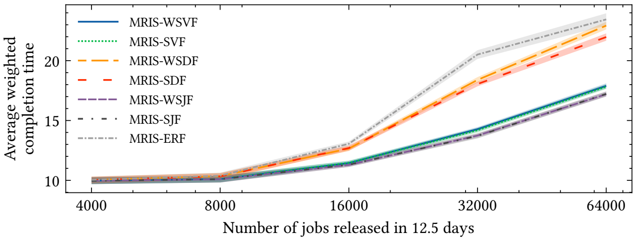

Figure 1: AWCT of MRIS under different sorting heuristics, with $M = 20$ machines.

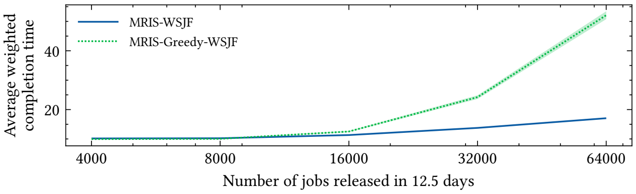

Figure 2: AWCT comparison between two knapsack subroutines, with $M = 20$ machines.

7.3 Sorting in MRIS

Since MRIS leverages PQ as a subroutine, there is freedom in how we sort jobs in each interval. In fact, the competitive ratio in Theorem 6.8 is independent of a sorting heuristic. We choose from the following popular heuristics and show the value used to sort jobs in non-decreasing order:

- (Weighted-)Smallest-Volume-First, (W)SVF: $v_j(/w_j)$
- (Weighted-)Shortest-Job-First, (W)SJF: $p_j(/w_j)$
- (Weighted-)Smallest-Demand-First, (W)SDF: $u_j(/w_j)$
- Earliest-Release-First (ERF): r_j

Figure 1 illustrates a clear hierarchy. ERF is weak because it simply schedules jobs as they arrive in a queue, ignoring processing times and demands. WSDF tries to pack jobs better than ERF but again ignores the time dimension. WSJF and WSVF perform the best, attributing to efficient packing in the time dimension. We further observe negligible difference between weighted and unweighted variants in this dataset due to the small range of priorities.

Therefore, for the remaining evaluations, we select WSJF when used as the sorting heuristic for the PQ subroutine in MRIS.

7.4 Choice of Knapsack Algorithm

In Section 6.1, we have found that CADP can use less additional knapsack space compared with the greedy approach in the worst case. We show this is true for the Azure dataset in the average case in Figure 2. MRIS, when using the greedy approach to solve knapsack (MRIS-Greedy), achieves 2% lower AWCT than dynamic programming for a small number of jobs (4000), but when the number of jobs increases, the greedy approach is over three times worse than CADP. Both algorithms meet the volume constraint for the maximum subset of jobs, but CADP is the better choice.

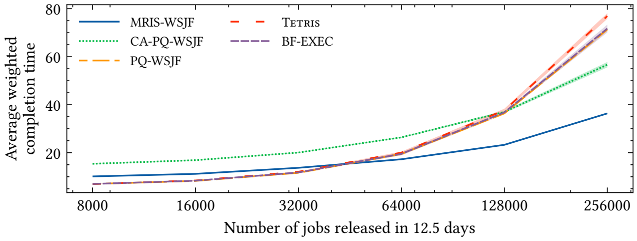

Figure 3: Effect of job arrival rate on AWCT, with $M = 20$ machines.

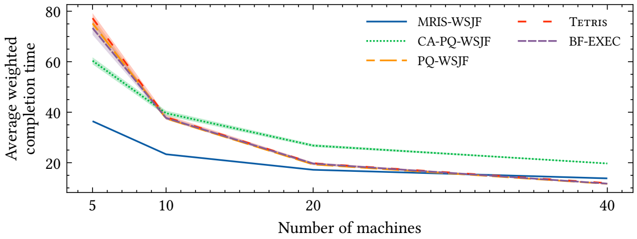

Figure 4: Effect of the number of machines on AWCT, with $N = 64000$ jobs.

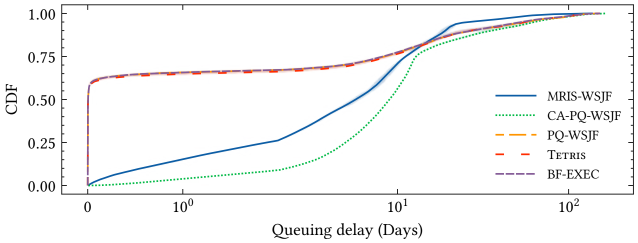

Figure 5: Queuing delay CDF of selected algorithms with $M = 20$ machines and $N = 64000$ jobs.

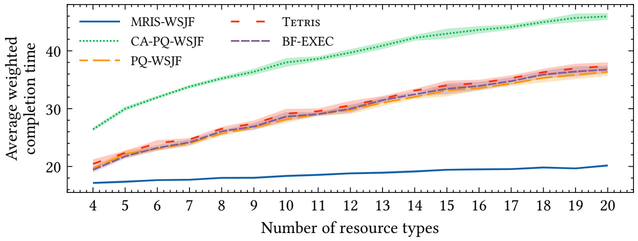

Figure 6: Effect of the number of resource types on AWCT, with $M = 20$ machines and $N = 64000$ jobs.

7.5 Performance Comparison

7.5.1 Weighted Average Completion Time. In Figure 3, we first observe that for a relatively small number of jobs, MRIS is outperformed by schedulers such as TETRIS, where the low number of jobs over the fixed 12.5-day release window is insufficient for MRIS to leverage all resources available in each interval. However, MRIS exceeds all others when job arrival increases. We observe that TETRIS, BF-EXEC, and PQ perform similarly across all job arrivals.

When there is a significant number of arrived jobs, 20 machines becomes insufficient and the other algorithms have similar performance to batching jobs and applying PQ with WSJF sorting, which serves as the worst-case reference. This suggests that most job processing occur after the last job has arrived, and the problem scenario moreso resembles an offline scheduling problem, where all job parameters are revealed to the scheduler at time zero. Nevertheless, MRIS can more effectively schedule jobs with short processing times, small demands, and large weights earlier than other schedulers, on average.

We also investigate the scaling of the number of machines in Figure 4. When a few machines are used to schedule a large number of jobs, resource contention is significant, and MRIS outperforms the others by achieving nearly half the AWCT compared to TETRIS. As more machines are available to relieve resource contention, PQ-WSVF is sufficient to schedule such jobs, whereas our algorithm is unable to maximize resource usage when using the interval construction. Figures 3 and 4 suggest that MRIS has a performance advantage under more heavily loaded scenarios.

7.5.2 Queuing Delay. To illustrate the source of the performance gains of MRIS, we plot the CDF of queuing delays for a selected number of jobs in Figure 5. In TETRIS, BF-EXEC, and PQ-WSJF, nearly 60% of the scheduled jobs have zero queuing delay, suggesting that many jobs that arrive are scheduled immediately in those algorithms. However, scheduling in this manner results in a sharp increase in the queuing delays for other jobs. This suggests that there are instances in which jobs are not treated fairly, as exemplified by Lemma 4.1. Observing MRIS, the queuing delay CDF increases more gradually. CA-PQ has the worst queuing delay because many jobs suffer from waiting until the last released job to start processing.

7.5.3 Synthetic Resource Scaling. Although the Azure dataset contains four distinct resource types, we investigate the scheduler performance when we augment the dataset to include more resource types. To achieve this, we let $\mathcal{D}$ be a job dataset. Then, for every new resource and for each job j, we uniformly sample a job $j' \sim \mathcal{D}$. j''s demand for this new resource is set to j's CPU demand. In this environment, we see that the other algorithms suffer significantly more than MRIS when the number of resources increases. For instance, in Figure 6, as the number of resources is increased from 4 to 20, the AWCT of TETRIS increases by 80%, whereas for MRIS, this is only 17%. We attribute this to MRIS's approach of better packing jobs through knapsack job selection and WSJF sorting.

7.5.4 Exercising Patience. Following the ideas of Section 4 we generate a synthetic input on one machine and multiple resources, where one job arrives at time zero that consumes 14 time units on the machine, not permitting other jobs to be run. Then, shortly after, nearly 2500 jobs arrive of random sizes and small, randomized job demands. PQ, TETRIS, and BF-EXEC all commit to the single large job prematurely, whereas MRIS exercises patience and schedules the smaller jobs first before committing to the single large job. The schedule is shown in Figure 7. Since nearly 2500 jobs suffer a delay of 14 time units, the AWCT of MRIS is nearly three times less than that of the other schedulers. Due to space constraints, the resource use over time is shown for the CPU resource.

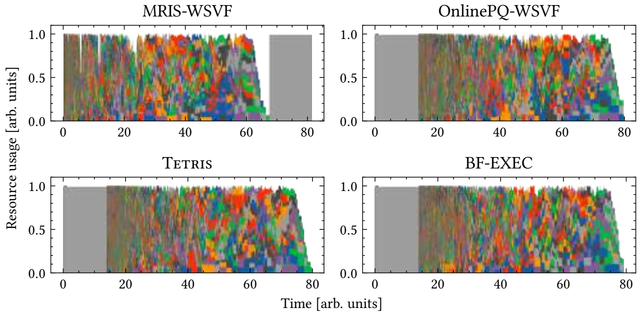

Figure 7: Schedules of algorithms on a specific synthetic input of $N = 2500$ jobs. Job's resource and processing demands are illustrated as non-contiguous rectangles. Rectangles with the same color across schedulers represent the same job.

8 CONCLUSION

In this work, we proposed Multi-Resource Interval Scheduling (MRIS), an online algorithm for scheduling jobs with multiple resource demands on multiple machines without the power of preemption. We studied the average weighted completion time, a popular objective in scheduling literature for fixed time horizons. To the best of our knowledge, our work is the first to study this problem under multiple resource constraints and in an online setting. We showed that MRIS is $8R(1+\epsilon)$-competitive for minimizing AWCT, and used trace-driven simulations to show that MRIS outperforms state-of-the-art schedulers, especially when the system is heavily loaded. As future work, one can be inspired by Bansal et al.'s algorithm, which has a sublinear dependence on the number of resources for the offline vector bin packing problem [1], to design a better makespan scheduler. A scheduler that jointly maximizes the total weight given a deadline can also be considered.

ACKNOWLEDGMENTS

This work was funded in part by the Department of National Defence under the IDEaS program.

REFERENCES

[1] Nikhil Bansal, Marek Eliáš, and Arindam Khan. 2016. Improved Approximation for Vector Bin Packing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*. Society for Industrial and Applied Mathematics, 1561–1579.

[2] Marin Bougeret, Pierre Francois Dutot, Klaus Jansen, Christina Otte, and Denis Trystram. 2010. Approximation Algorithms for Multiple Strip Packing. In *Approximation and Online Algorithms (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 37–48.

[3] Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Robert I. Davis. 2017. On the Pitfalls of Resource Augmentation Factors and Utilization Bounds in Real-Time Scheduling. In *Proceedings of 29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. 9:1–9:25.

[4] Mukoe Cheong, Hyunsung Lee, Ikjun Yeom, and Honguk Woo. 2019. SCARL: Attentive Reinforcement Learning-Based Scheduling in a Multi-Resource Heterogeneous Cluster. *IEEE Access* 7 (2019), 153432–153444.

[5] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. 2017. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review* 24 (May 2017), 63–79.

[6] Leah Epstein and Rob van Stee. 2005. Optimal Online Algorithms for Multidimensional Packing Problems. *SIAM J. Comput.* 35, 2 (Jan. 2005), 431–448.

[7] Kyle Fox and Madhukar Korupolu. 2013. Weighted Flowtime on Capacitated Machines. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 129–143.

[8] Michael R. Garey and Ronald L. Graham. 1975. Bounds for Multiprocessor Scheduling with Resource Constraints. *SIAM J. Comput.* 4, 2 (June 1975), 187–200.

[9] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*. 323–336.

[10] Ronald L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.* 17, 2 (March 1969), 416–429.

[11] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 455–466.

[12] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E. Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. 2020. Protean: VM Allocation Service at Scale. In *Proceedings of 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. 845–861.

[13] Leslie A. Hall, David B. Shmoys, and Joel Wein. 1996. Scheduling to minimize average completion time: off-line and on-line algorithms. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms (SODA '96)*. Society for Industrial and Applied Mathematics, USA, 142–151.

[14] Oscar H. Ibarra and Chul E. Kim. 1975. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *J. ACM* 22, 4 (Oct. 1975), 463–468.

[15] Sungjin Im, Janardhan Kulkarni, Benjamin Moseley, and Kamesh Munagala. 2016. A Competitive Flow Time Algorithm for Heterogeneous Clusters Under Polytope Constraints. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 60)*. 10:1–10:15.

[16] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. 2017. Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints. *J. ACM* 65, 1 (Dec. 2017), 3:1–3:33.

[17] Sungjin Im, Mina Naghshnejad, and Mukesh Singhal. 2016. Scheduling jobs with non-uniform demands on multiple servers without interruption. In *Proceedings of The 35th Annual IEEE International Conference on Computer Communications (INFOCOM 2016)*. 1–9.

[18] Bala Kalyanasundaram and Kirk Pruhs. 2000. Speed is as powerful as clairvoyance. *J. ACM* 47, 4 (July 2000), 617–643.

[19] Eugene L. Lawler. 1977. Fast approximation algorithms for knapsack problems. In *Proceedings of 18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 206–213.

[20] Chang-Gun Lee, Hoosun Hahn, Yang-Min Seo, Sang Lyul Min, Rhan Ha, Seongsoo Hong, Chang Yun Park, Minsuk Lee, and Chong Sang Kim. 1998. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Trans. Comput.* 47, 6 (June 1998), 700–713.

[21] MohammadJavad NoroozOliaee, Bechir Hamdaoui, Mohsen Guizani, and Mahdi Ben Ghorbel. 2014. Online multi-resource scheduling for minimum task completion time in cloud servers. In *Proceedings of 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 375–379.

[22] David Pisinger. 1999. Linear Time Algorithms for Knapsack Problems with Bounded Weights. *Journal of Algorithms* 33, 1 (Oct. 1999), 1–14.

[23] Konstantinos Psychas and Javad Ghaderi. 2017. On Non-Preemptive VM Scheduling in the Cloud. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 2 (Dec. 2017), 35:1–35:29.

[24] Konstantinos Psychas and Javad Ghaderi. 2018. Randomized Algorithms for Scheduling Multi-Resource Jobs in the Cloud. *IEEE/ACM Transactions on Networking* 26, 5 (Oct. 2018), 2202–2215.

[25] Uwe Schwiegelshohn, Andrei Tchernykh, and Ramin Yahyapour. 2008. Online scheduling in grids. In *Proceedings of 2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–10.

[26] John Turek, Uwe Schwiegelshohn, Joel L. Wolf, and Philip S. Yu. 1994. Scheduling parallel tasks to minimize average response time. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms (SODA '94)*. USA, 112–121.

[27] Wei Wang, Ben Liang, and Baochun Li. 2015. Multi-Resource Fair Allocation in Heterogeneous Cloud Computing Systems. *IEEE Transactions on Parallel and Distributed Systems* 26, 10 (Oct. 2015), 2822–2835.

[28] Gerhard J. Woeginger. 1997. There is no asymptotic PTAS for two-dimensional vector packing. *Inform. Process. Lett.* 64, 6 (Dec. 1997), 293–297.

[29] Huanle Xu, Yang Liu, and Wing Cheong Lau. 2023. Multi Resource Scheduling with Task Cloning in Heterogeneous Clusters. In *Proceedings of the 51st International Conference on Parallel Processing (ICPP '22)*. 1–11.

[30] Bo Yin, Yu Cheng, Lin X. Cai, and Xianghui Cao. 2017. Online SLA-Aware Multi-Resource Allocation for Deadline Sensitive Jobs in Edge-Clouds. In *Proceedings of 2017 IEEE Global Communications Conference (GLOBECOM 2017)*. 1–6.

[31] Rui Zhang, Kui Wu, Minming Li, and Jianping Wang. 2016. Online Resource Scheduling Under Concave Pricing for Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems* 27, 4 (April 2016), 1131–1145.

[32] Zizhan Zheng and Ness B. Shroff. 2016. Online multi-resource allocation for deadline sensitive jobs with partial values in the cloud. In *The 35th Annual IEEE International Conference on Computer Communications (INFOCOM 2016)*. 1–9.