Online Two-dimensional Load Balancing

Ilan Reuven Cohen
Centrum Wiskunde & Informatica, Netherlands
ilanrcohen@gmail.com

Sungjin Im
Department of Computer Science and Engineering, University of California Merced, USA
sim3@ucmerced.edu

Debmalya Panigrahi
Department of Computer Science, Duke University, USA
debmalya@cs.duke.edu

Abstract

In this paper, we consider the problem of assigning 2-dimensional vector jobs to identical machines online so to minimize the maximum load on any dimension of any machine. For arbitrary number of dimensions d, this problem is known as vector scheduling, and recent research has established the optimal competitive ratio as $O\left(\frac{\log d}{\log \log d}\right)$ (Im *et al.* FOCS 2015, Azar *et al.* SODA 2018). But, these results do not shed light on the situation for small number of dimensions, particularly for $d = 2$ which is of practical interest. In this case, a trivial analysis shows that the classic list scheduling greedy algorithm has a competitive ratio of 3. We show the following improvements over this baseline in this paper:

- We give an improved, and tight, analysis of the list scheduling algorithm establishing a competitive ratio of $8/3$ for two dimensions.
- If the value of OPT is known, we improve the competitive ratio to $9/4$ using a variant of the classic best fit algorithm for two dimensions.
- For any fixed number of dimensions, we design an algorithm that is provably the best possible against a fractional optimum solution. This algorithm provides a proof of concept that we can simulate the optimal algorithm online up to the integrality gap of the natural LP relaxation of the problem.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, scheduling, multi-dimensional

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.54

Funding *Ilan Reuven Cohen*: Supported by the ERC Consolidator Grant 617951.
Sungjin Im: Supported in part by NSF grants CCF-1409130, CCF-1617653, and CCF-1844939.
Debmalya Panigrahi: Supported in part by NSF grants CCF-1535972, CCF-1955703, an NSF CAREER Award CCF-1750140, and the Indo-US Virtual Networked Joint Center on Algorithms under Uncertainty.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 54; pp. 54:1–54:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the online load balancing problem, the goal is to allocate n jobs appearing online on a set of m identical machines so as to minimize the maximum load on any machine (called *makespan*). This problem was introduced in the 1960s by Graham [26, 27], who gave the list scheduling algorithm that assigns each arriving job to the machine with minimum load, and achieves a competitive ratio[1] of 2.[2] Since then, there has been a long line of work that aims to improve this constant below 2, both when the optimal value OPT is unknown [10, 37, 1, 19, 18, 11, 25, 28, 2], and when OPT is known [9, 40, 4, 38, 39, 21, 20, 12]. The current record is a competitive ratio of 1.916 due to Albers [2] for unknown OPT, and 1.5 due to Bohm *et al.* [12] for known OPT.

Recent research has further expanded the scope of this problem to vector jobs that have multiple dimensions, the resulting problem being called *vector scheduling* [15, 7, 43, 29, 8, 30]. As earlier, the goal is to minimize the makespan of the assignment, which now represents the maximum load across all machines *and all dimensions*. This problem arises in data centers where jobs with multiple resource requirements have to be allocated to machine clusters to make efficient use of limited resources such as CPU, memory, and network bandwidth [24, 44, 41, 17, 31, 32].

It is now known that the right dependence of the competitive ratio for vector scheduling on the number of dimensions d is $\Theta(\log d/\log\log d)$ [29]. While this gives a satisfactory answer when the number of dimensions is large, in the practical context, the number of dimensions is usually small since they represent distinct computational resources. In particular, the majority of the systems scheduling literature (e.g., see [24] and follow-up papers) considers only two resources, CPU and memory, since they often tend to be the most critical bottleneck resources. Unfortunately, the existing bounds for vector scheduling do not shed any light on this case since we are interested in optimizing the constant in the competitive ratio. In this paper, we initiate the study of the *online 2-dimensional scheduling* problem, or 2DSCHED in short.

1.1 Results and Techniques

Baseline. Graham's list scheduling algorithm can be naturally extended to $d > 1$ dimensions by assigning each job to the machine that minimizes the makespan after the assignment. This algorithm has a competitive ratio of 3 for $d = 2$ (and $d + 1$ for general d). To see this, assume wlog the optimum makespan OPT $= 1$ by scaling. Since the average load on each dimension is at most OPT, it follows that there is always some machine where the sum of loads on its two dimensions is at most 2. Consequently, this machine has a load of at most 2 on each of its dimensions. Now, note that the load of any single job cannot exceed OPT on any dimension; hence, the maximum load on a machine after the greedy assignment of a job cannot exceed 3.

Unfortunately, despite the aforementioned recent progress, no existing algorithms are known to have a competitive ratio better than 3. Thus, our goal is to break the 3-competitive ratio barrier for the problem. This requires a new approach since the existing analytical methods are based on potential functions, concentrations, or volume bounds, and they all seem to inevitably lose a considerable constant factor in the competitive ratio.

[1] The *competitive ratio* of an online minimization problem is the maximum ratio between the objective of the algorithm and that of an (offline) optimal solution, see e.g., [13].

[2] The competitive ratio is actually $2 - 1/m$ for m machines, but we will ignore $o(1)$ terms throughout since we consider instances of arbitrary problem size in this paper.

A Novel Analytical Technique: Characterizing Reachable States

Our new approach is to directly characterize the set of reachable states of the algorithm. To illustrate our approach, let us take a close look at the above analysis of list scheduling. For the analysis to be tight, a configuration (machine loads) must be created where half the machines have a load of (roughly, ignoring lower order terms) $(2, 0)$ and the remaining half have a load of $(0, 2)$. If a job of load $(1, 1)$ now arrives, then the maximum load will increase to 3 no matter where the job is assigned. But, do we ever create such imbalance in the configurations of the machines?

To rule out such states/configurations, we need to define the set of reachable states of the algorithm. Our first contribution is to develop a general framework that allows us to characterize the set of reachable states of not only the greedy algorithm, but of a much larger class of algorithms that we call *priority-based algorithms*. Roughly speaking, these are algorithms where the newly arriving job is assigned to minimize a "disutility function" that maps the current load of the machines (i.e., the current state) and the load of the current job to a real number. For such algorithms, our main observation is that if m machines come to have load vectors $c = (c_1, c_2, ..., c_m)$ – meaning that this configuration is reachable – then any pair (c_i, c_j) is a reachable state for the same algorithm on just two machines. Furthermore, we identify a specific pair (c_i, c_j) that captures the essential characteristics of the algorithm under consideration.

Using this framework, we show that the greedy makespan minimization algorithm that we described above is $8/3$-competitive – and our analysis is tight. We defer the lower bound to the full version of the paper.

▶ **Theorem 1.** *(Section 4) There exists a priority-based algorithm that is $8/3$-competitive for the* 2DSCHED *problem with unknown* OPT*. Furthermore, this analysis is tight.*

If we know the value of OPT, then we obtain a better competitive ratio of 2.25 by using a different algorithm that explicitly minimizes the difference of the loads on the two dimensions without violating the preset threshold $\alpha \cdot$ OPT, where $\alpha = 2.25$ is the desired competitive ratio. Again, this analysis is tight, the proof of which we defer to the full version of the paper. This "balance algorithm" can be thought of as a generalization of the popular *best fit* strategy used in bin packing (see [33, 35] for one-dimensional bin packing).

▶ **Theorem 2.** *(Section 5) There exists a priority-based algorithm that is 2.25-competitive for the* 2DSCHED *problem with known* OPT*. Furthermore, this analysis is tight.*

Recall that the minimum makespan problem for $d = 1$ has been widely studied for both the known and unknown OPT scenarios, and our results obtain corresponding bounds for the 2DSCHED problem.

As further evidence of the generality of our analysis framework, we also analyze a natural extension of the popular *first fit* rule used for bin packing problems. (The reader is referred to [23, 6] for multi-dimensional first fit bin packing and [16] for a full survey on one-dimensional first fit bin packing.) In this algorithm, given a target competitive ratio, the algorithm assigns a new job to the first bin that does not violate the competitiveness guarantee. This can be implemented as stated if OPT is known, and has a tight competitive ratio of 2.5. (The proof of the next theorem is deferred to the full version of the paper.)

▶ **Theorem 3.** *The first fit algorithm is 2.5-competitive for the* 2DSCHED *problem with known* OPT*. Furthermore, this analysis is tight.*

We also show that if the first fit algorithm is suitably augmented with a framework for guessing the value of OPT and adjusting this guess over time, then it has a competitive ratio better than the naïve bound of 3 for unknown OPT. (The proof of the next theorem is also deferred to the full version of the paper.)

▶ **Theorem 4.** *The first fit algorithm is* 2.89*-competitive for the* 2DSCHED *problem with unknown* OPT.

While we only showcase the power of our framework by giving tight analyses of the algorithms described above, we believe that our framework has the potential to find more applications. This is because it reduces characterizing the reachable states for an arbitrary number of machines to those for only two machines, making the search space of the worst-case assignment much more tractable from an analytical perspective. In fact, our framework is easily extendable to arbitrary d, so that we only need to consider reachable states pretending that there are only d machines. While we currently know how to analytically characterize the reachable states only when $d = 2$, and therefore the results in this paper are only for this case, it is plausible (and an interesting direction of future work) to further extend such a characterization to higher dimensions analytically and/or numerically using the fact that the number of available machines is small. In that case, our framework would be useful in providing results for online vector scheduling in $d > 2$ dimensions, e.g., in cases of three or four dimensions that are also of practical interest.

A Near-optimal Algorithm

We now switch our attention to a different type of algorithmic result. Note that the competitive ratio of all the known algorithms for $d \geq 2$ are based on their comparison against the fractional optimum. That is, as long as the total load vector is $m \cdot \vec{1}$, and each job load vector is at most $\vec{1}$, an α-competitive algorithm produces a schedule where the load vector on each machine is at most $\alpha \cdot \vec{1}$. Note that when $d = 1$, the competitive ratio 2 is also obtained against the fractional optimum and even the best competitive ratio 1.916 against the actual optimum is not far from 2.

Our next result is to give an online algorithm whose competitive ratio nearly matches the best one can hope for against the fractional optimum for any fixed d. Here our high-level approach is as follows. We first use a variant of the algorithm in [29] to assign jobs to groups of machines, ensuring that every group receives at most $1 + \epsilon$ times its share of the load in an optimal fractional solution. Then, we assign jobs to machines within each group. We differentiate between "big" jobs and "small" jobs in this assignment. For the big jobs, we use discretization to bound the number of job types, and then use an optimal decision tree to make the actual assignments. Note that the optimal decision tree can be found *offline* for every possible job arrival pattern since the total number of big jobs in a group is small, and the one that matches the online sequence can be pressed into service in the online algorithm. To assign small jobs using the decision tree, we batch and encapsulate small jobs of similar load vectors into *bin* vectors. To enable this online, we pre-allocate some bin vectors. Thus, we can effectively reduce the problem of assigning small jobs to the scalar bin packing using pre-allocation and the decision tree.

▶ **Theorem 5.** *(Section 6) For any* $d \geq 1$ *and* $\epsilon > 0$*, assuming that the value of the optimum makespan*[3] *is known a priori, there exists a deterministic online algorithm for the online vector load balancing problem whose competitive ratio is* $(1 + \epsilon)c^*_d$*, where* c^*_d *is the best competitive ratio one can hope for against the fractional optimum. Furthermore, the running time of the algorithm is polynomial in* n *for any fixed* d, ϵ*.*[4] *(For a more formal statement of this result, see Definition 23 and Theorem 24.)*

Before closing this section, we note the contrast between the first set of results based on the new analysis framework, and the last result that yields the nearly best competitive ratio against

[3] More accurately, the value of the fractional optimum makespan.

[4] More precisely, the running time is polynomial in n and $(d/\epsilon)^{d^{(d/\epsilon)^{O(d)}}}$.

the fractional optimum. While the near optimality of the last solution is attractive, the first set of algorithms are much more practical since the complexity of obtaining an optimal decision tree is likely to be prohibitive for the last algorithm. Furthermore, the last result does not give a numerical performance guarantee, unlike the first set of algorithms. Indeed, these two sets of results complement each other, and cumulatively provide the first insights into the 2DSCHED problem.

1.2 Related Work

The online load balancing problem for 1-dimensional jobs has had a long history. It was introduced by Graham in the 1960s [26, 27], who gave the list scheduling algorithm with a competitive ratio of 2. In the last three decades, there have been a series of results for improving the competitive ratio below 2 and obtaining lower bounds on the competitive ratio [10, 37, 1, 19, 18, 11, 25, 28, 2]. The best algorithm known is a 1.916-competitive algorithm due to Albers [2]. These results address the situation where the online algorithm does not know the value of OPT. Azar and Regev [9] introduced the problem of online load balancing when OPT is known, and called this problem *bin stretching*. For bin stretching, a series of results [40, 4, 38, 39, 21, 20, 12] have led to a 1.5-competitive algorithm due to Böhm *et al.* [12].

Recent research has expanded the scope of this problem to vector jobs, the resulting problem being called vector scheduling. Matching upper and lower bounds of $O(\log d/\log\log d)$ have been derived for d dimensions [15, 7, 43, 29]. Note that since the competitive ratio is super-constant, OPT can be assumed to be known by a standard guess and double trick. All these results are for an arbitrary number of machines and jobs. There is a large literature on variations, generalizations, and special cases of these problems, such as optimizing norms other than makespan, considering non-identical machines, focusing on a small constant number of machines, handling only jobs of small size, etc. that we omit here for brevity. The reader is referred to several excellent surveys on the topic, e.g., by Azar [5], Sgall [46, 47], Pruhs, Sgall, and Torng [45], Albers [3], etc.

The online load balancing problem is also related to the online bin packing problem, where the capacity of every machine is fixed and the goal is to minimize the number of machines used. For a single dimension, this problem has been studied since the work of Johnson in the 1970s; see, e.g., [34, 36] and surveys [22, 48]. For vector jobs, the problem was introduced by Garey *et al.* [23] and has been extensively studied in the last few years [7, 6, 8].

We note that some results of a flavor similar to Theorem 5 are known for other scheduling problems. Specifically, Lübbecke et al. [42] showed online algorithms of competitive ratios arbitrarily close to the optimum for the objective of minimizing total weighted completion time and its generalizations on unrelated machines. Various types of priority-based algorithms have been extensively studied for various scheduling problems. See [14] for the relevant pointers and follow-up works.

Roadmap

We present the general framework for analyzing priority based algorithms in Section 3 and use it to analyze the greedy algorithm in Section 4 for unknown OPT and the balance algorithm in Section 5 for known OPT. Finally, we present the near-optimal algorithm against the fractional optimum in Section 6.

2 Preliminaries

In this paper, we focus on the online 2-dimensional scheduling problem, or 2DSCHED in short. In this problem, a set of n jobs $\mathbf{V}$, indexed by $j \in [n]$, and represented by 2-dimensional non-negative vectors $v_j = (v_j(1), v_j(2))$ arrive in an online sequence. On arrival, a job must be assigned to one of a given set of m identical machines M, indexed by $i \in [m]$. The goal of the algorithm is to minimize the *makespan*, which is defined as the maximum load on any dimension of any machine. Formally, for any machine i, let V_i^j denote the set of vectors assigned to this machine after the arrival of the j'th vector. Then, we say that machine i's *configuration* is given by $c_i^j = \sum_{v_{j'} \in V_i^j} v_{j'}$, which is (we may omit the superscript j if it is clear from the context):

$$(c_i^j(1), c_i^j(2)) = \left(\sum_{v_{j'} \in V_i^j} v_{j'}(1), \sum_{v_{j'} \in V_i^j} v_{j'}(2) \right).$$

For any value k, we say $c_i \leq \vec{k}$ if $c_i(1) \leq k$ *and* $c_i(2) \leq k$; otherwise, we say $c_i \nleq \vec{k}$ if at least one of these inequalities are violated, i.e., if $c_i(1) > k$ *or* $c_i(2) > k$. Analogously, we define $c_i \geq \vec{k}$ if $c_i(1) \geq k$ *and* $c_i(2) \geq k$; otherwise, we say $c_i \ngeq \vec{k}$.

Let us denote the optimal offline configuration by OPT; overloading notation, let OPT also represent the makespan of this configuration. The online algorithm is said to be α-competitive if $c_i^n \leq \vec{a}$ for all $i \in [m]$, where $a = \alpha \cdot \text{OPT}$. We show two sets of results, the first when the value of OPT is known and the second when OPT is unknown to the algorithm. Note that if OPT is unknown, then its value is not used in the definition of the algorithm. Nevertheless, for the sake of the analysis, we normalize and set $\text{OPT} = 1$, which also implies that for all job vectors $v_j \in V$, we have $v_j(1), v_j(2) \in [0, 1]$, and $\sum_{j \in [n]} v_j / m \leq \vec{1}$. For convenience, we call the first coordinate the *left* coordinate, and the second coordinate the *right* coordinate. We call a job a *left vector* if its left coordinate is larger than or equal to its right coordinate, and a *right vector* otherwise.

3 Priority-based Algorithms

In this section, we give a framework to analyze a large class of algorithms that prioritize machines based on their current load. More specifically, such an algorithm computes a certain disutility for each machine only using its current load and the arriving job's load and assigns it to a machine with the least disutility. Thus, this class of algorithms are completely determined by the disutility function: $u : (c, g) \to [0, \infty]$ where $c \in [0, \infty)^2$ represents a machine's current load vector, and $g \in [0, \infty)^2$ a job's load vector. Formally, given a set $[m]$ of machines, the algorithm $\textsc{Priority}(u)$ assigns job j to machine $i^* := \arg\min_{i \in [m]} u(c_i^{j-1}, v_j)$ breaking ties arbitrarily but consistently. Here, as mentioned before, c_i^{j-1} denotes machine i's load just before assigning job j. After assigning job j, we update $c_{i^*}^j = c_{i^*}^{j-1} + v_j$ while keeping $c_i^j = c_i^{j-1}$ for all $i \neq i^*$. If $u(c_i^{j-1}, v_j) = \infty$ for all $i \in [m]$, then $\textsc{Priority}(u)$ declares failure.

3.1 Analysis Framework: Zooming in on Jobs Assigned to Two Machines

Now we present our general framework to analyze the above type of priority-based algorithms. The key to this framework is to define the set of reachable configurations.

▶ **Definition 6.** *We say that an ordered tuple $C = (c_1, c_2, \ldots, c_m)$, which we call a configuration, where $c_i \in [0, \infty)^2$, is reachable by $\textsc{Priority}(u)$ if machines have load vectors $c_1, c_2, \ldots, c_m$ after $\textsc{Priority}(u)$ assigns some sequence of jobs $[n]$ to machines $[m]$, such that $||v_j||_\infty \leq 1$ for all $j \in [n]$ and $\sum_{j \in [n]} v_j \leq m \cdot \vec{1}$. The set of reachable configurations is denoted by $\mathcal{R}_m(u)$.*

Unfortunately, it seems extremely challenging to characterize the reachable configurations for m machines in general. Our key observation is that the priority-based choices made by our algorithms allows us to focus on the loads of only two machines.

▶ **Observation 7.** *For any disutility function u and configuration $C = (c_1, c_2, \dots, c_m) \in \mathcal{R}_m(u)$, and for any pair $i \neq j \in [m]$, we have $(c_i, c_j) \in \mathcal{R}_2(u)$.*

Proof. For notional convenience, say machines i and j have load vectors c_i and c_j, respectively. Consider the jobs that are assigned to machines i and j. If we assign those jobs to machines i and j pretending that no other machines exist, the two machines i and j each would get assigned exactly the same set of jobs. This is because PRIORITY(u) prioritizes machines only based on their current respective load vector and the arriving job's load vector. Thus, we have shown $(c_i, c_j) \in \mathcal{R}_2(u)$. ◀

Now, the looming question is which pair of machines we should focus on. If α is the target competitive ratio we want to establish, we would like to focus on critical machines, i.e., where one of the coordinates has a load exceeding $\alpha - 1$ since they cannot accommodate a job of load vector $\vec{1}$. Also, we would like to focus on a pair where the 'average' load between the two dimensions is not so high to draw a contradiction—more precisely, a convex combination of the load vectors of the two machines should be capped by $\vec{1}$. To denote such a pair, we will often use the notation $p = (p_f, p_s)$, where $p_f, p_s \in [0, \infty)^2$,

▶ **Definition 8.** *For an unordered pair of configuration $p = (p_f, p_s)$, we define*

- *$p \in \mathcal{L}(\alpha)$ iff $p_f + \vec{1} \not\leq \vec{\alpha}$ and $p_s + \vec{1} \not\leq \vec{\alpha}$, and we say p is* overloaded*; and*
- *$p \in \mathcal{F}$ iff for all $\lambda \in [0, 1]$, we have $\lambda \cdot p_f + (1 - \lambda) \cdot p_s \not\leq \vec{1}$, and we say p is* overflown.

The next lemma shows that there will always be at least one pair of configurations that has not overflown.

▶ **Lemma 9.** *For any $C = (c_1, c_2, \dots, c_m) \in \mathcal{R}_m(u)$ such that $c_i \neq \vec{0}$ for all $i \in [m]$, there exist $k \neq \ell \in [m]$ such that $(c_k, c_\ell) \notin \mathcal{F}$.*

Proof. Let $q = \frac{\sum_{i \in [m]} c_i}{m}$; note $q \leq \vec{1}$ since $C \in \mathcal{R}_m(u)$. Clearly, q is in the convex hull of the vectors, $c_1 \dots, c_m$. However, the convex hull doesn't include $\vec{0}$. Since the convex hull is in two-dimensional space, this means there exists $\gamma \in (0, 1]$, such that $\gamma \cdot q$ is on the segment (c_k, c_ℓ) for some $k, \ell \in [m]$. Thus, there exists $\lambda \in [0, 1]$ such that

$$\lambda \cdot c_k + (1 - \lambda) \cdot c_\ell = \gamma \cdot q.$$

As $q \leq \vec{1}$, we have $(c_k, c_\ell) \notin \mathcal{F}$. ◀

The following observation is immediate from the definition of $\mathcal{L}(\alpha)$ and $\mathcal{F}$, it will be useful to first enlist the different cases in terms of these individual coordinate values.

▶ **Observation 10.** *For any $\alpha > 2$, if $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$, then we have:*
- either $p_f(2), p_s(1) > 1$ and $p_f(1), p_s(2) < 1$;
- or, $p_f(1), p_s(2) > 1$ and $p_f(2), p_s(1) < 1$.

If the algorithm PRIORITY(u) reaches a state where no machine can accommodate another job of load $\vec{1}$, then using Lemma 9, we can find a pair of configurations in $\mathcal{L}(\alpha) \setminus \mathcal{F}$. Then, using the facts that the pair is overloaded yet not overflown, we can determine the sign of a certain function $V(p_f, p_s)$ defined on the configuration's load vectors; this function will be useful to draw a contradiction later. Formally, for a pair of configuration (p_f, p_s), define

$$V(p_f, p_s) := \frac{p_f(2) \cdot (p_s(1) - 1) + p_s(2) \cdot (1 - p_f(1))}{p_s(1) - p_f(1)} - 1 \tag{1}$$

▶ **Lemma 11.** *For $\alpha > 2$, if $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$, then we have $V(p_f, p_s) \leq 0$.*

Proof. Since $p_f + 1, p_s + 1 \nleq \vec{\alpha}$ and $\alpha > 2$, we can assume wlog that $p_f(2) > 1$; the other case $p_f(1) > 1$ can be handled similarly. Then, we must have

$$p_s(2) < 1 \text{ and } p_s(1) > 1 \text{ and } p_f(1) < 1,$$

since $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$; see Observation 10. Define

$$f(\lambda, k) := \lambda \cdot p_f(k) + (1-\lambda) \cdot p_s(k).$$

Since $(p_f, p_s) \notin \mathcal{F}$, there must exist $\lambda^* \in [0,1]$ such that $f(\lambda^*, 1), f(\lambda^*, 2) \leq 1$. Observe $f(\lambda, k)$ is monotonically decreasing in λ when $k = 1$, and monotonically increasing when $k = 2$. For

$$\tilde{\lambda} = \frac{p_s(1) - 1}{p_s(1) - p_f(1)} \in [0,1], \text{ i.e., } 1 - \tilde{\lambda} = \frac{1 - p_f(1)}{p_s(1) - p_f(1)},$$

we have $f(\tilde{\lambda}, 1) = 1$. Since $f(\lambda, 1)$ is monotonically decreasing, $\lambda^* \geq \tilde{\lambda}$. Since $f(\lambda, 2)$ is monotonically increasing, we have

$$f(\tilde{\lambda}, 2) = \frac{p_f(2) \cdot (p_s(1) - 1) + p_s(2) \cdot (1 - p_f(1))}{p_s(1) - p_f(1)} \leq f(\lambda^*, 2) \leq 1,$$

as desired. ◀

Having established the sign of $V(p_f, p_s)$, we now observe that it is an increasing function in any of the coordinates of the two configurations p_f, p_s.

▶ **Observation 12.** *For $\alpha > 2$ and $p = (p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ we have*

$$\left(\frac{\partial V(p_f, p_s)}{\partial p_f(1)}, \frac{\partial V(p_f, p_s)}{\partial p_f(2)}, \frac{\partial V(p_f, p_s)}{\partial p_s(1)}, \frac{\partial V(p_f, p_s)}{\partial p_s(2)} \right) > \vec{0}.$$

Proof. By taking partial derivatives on $p_f(1), p_f(2), p_s(1), p_s(2)$ respectively, we have

$$\begin{aligned}
&\left(\frac{\partial V(p_f, p_s)}{\partial p_f(1)}, \frac{\partial V(p_f, p_s)}{\partial p_f(2)}, \frac{\partial V(p_f, p_s)}{\partial p_s(1)}, \frac{\partial V(p_f, p_s)}{\partial p_s(2)} \right) \\
&= \left(\frac{(p_f(2) - p_s(2)) \cdot (p_s(1) - 1)}{(p_f(1) - p_s(1))^2}, \frac{1 - p_s(1)}{p_f(1) - p_s(1)}, \frac{(p_f(2) - p_s(2)) \cdot (1 - p_f(1))}{(p_f(1) - p_s(1))^2}, \frac{p_f(1) - 1}{p_f(1) - p_s(1)} \right) \\
&> \vec{0},
\end{aligned}$$

where the last inequalities follow from Observation 10. ◀

▶ **Corollary 13.** *For any $\alpha > 2$, if $(p_f, p_s), (p'_f, p'_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$, and $p_f \geq p'_f$ and $p_s \geq p'_s$, then we have $V(p_f, p_s) \geq V(p'_f, p'_s)$.*

We now have all the pieces for the refined analysis of $\textsc{Priority}(u)$. Suppose we want to show $\textsc{Priority}(u)$ is α-competitive. Towards this end, it is sufficient to show that if $C = (c_1, c_2, \ldots, c_m) \in \mathcal{R}_m(u)$, then we have $c_i + \vec{1} \leq \vec{\alpha}$ for some $i \in [m]$. For the sake of contradiction, suppose not. Then, by Lemmas 9 and 11, we have $(c_k, c_\ell) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ for some $k, \ell \in [m]$, and $V(c_k, c_\ell) \leq 0$. Further, by Observation 7, we know $(c_k, c_\ell) \in \mathcal{R}_2(u)$. This leads to the following lemma.

▶ **Lemma 14.** *If $\mathcal{R}_2(u) \cap \mathcal{L}(\alpha) \setminus \mathcal{F} = \emptyset$, then $\textsc{Priority}(u)$ is α-competitive.*

Note that this lemma allows us to analyze $\textsc{Priority}(u)$ pretending that there are only *two* machines. Now showing the condition $\mathcal{R}_2(u) \cap \mathcal{L}(\alpha) \setminus \mathcal{F} = \emptyset$ of the lemma depends on α and the disutility function u governing $\textsc{Priority}(u)$.

4 Greedy Algorithm: Unknown OPT

In this section we consider the natural algorithm that assigns an arriving job to the machine yielding the minimum makespan. We recover this algorithm by setting the disutility function u to be the following:

$$\textsc{Max}(c, g) = ||c + g||_\infty \tag{2}$$

We call this algorithm $\textsc{Priority}(\textsc{Max})$. Our goal in this section is to prove the following theorem.

▶ **Theorem 15** (Upper Bound of Theorem 1). $\textsc{Priority}(\textsc{Max})$ *is* $8/3$*-competitive for the* 2DSCHED *problem.*

To show Theorem 15, we will set $\alpha = 8/3$ and use Lemma 14. We begin with an easy observation, which immediately follows from $||v_j||_\infty \le 1$ for all jobs j. (The latter is a consequence of normalizing OPT in the analysis, and not an assumption on the input.)

▶ **Observation 16.** *For any* $p = (p_s, p_f) \in \mathcal{R}_2(\textsc{Max})$*, we have* $|\ ||p_s||_\infty - ||p_f||_\infty| \le 1$.

It is straightforward to show $\textsc{Priority}(\textsc{Max})$ is 3-competitive using this observation. To obtain a tighter bound, we will show the following:

▶ **Lemma 17.** *For* $\alpha = 8/3$*, we have* $(\mathcal{R}_2(\textsc{Max}) \cap \mathcal{L}(\alpha)) \setminus \mathcal{F} = \emptyset$.

Note that Lemma 17 implies Theorem 15 by applying it to Lemma 14. So, the rest of this section is devoted to proving Lemma 17.

For a pair of configurations (p_f, p_s), define

$$H_1(p_f, p_s) := p_s(2) + p_f(1) - p_s(1) + 1$$
$$H_2(p_f, p_s) := p_s(2) + p_f(1) - p_f(2) + 1$$

▶ **Lemma 18.** *For a pair of configurations* $p = (p_f, p_s)$*, we have* $p \notin \mathcal{R}_2(\textsc{Max})$ *if*

$$\min\{H_1(p_f, p_s), H_2(p_f, p_s), H_1(p_s, p_f), H_2(p_s, p_f)\} < 0.$$

Proof. Assume for the sake of contraction that there exists $(p_f, p_s) \in \mathcal{R}_2(\textsc{Max})$ such that the minimum is non-negative. Further, assume that (p_f, p_s) is one among such configurations that is reachable by the minimum number of jobs assigned. Clearly, $(p_f, p_s) \ne ((0,0),(0,0))$. Since (p_f, p_s) is unordered, assume wlog that there exist c, g such that $p_f = c + g$ and $(c, p_s) \in \mathcal{R}_2(\textsc{Max})$ and $g \le \vec{1}$ can be assigned to (a machine of load) c according to $\textsc{Priority}$ $(\textsc{Max})$, meaning $||c + g||_\infty \le ||p_s + g||_\infty$. We consider two cases.

Case 1: $H_1(p_f, p_s) < 0$; this is symmetric to $H_2(p_s, p_f) < 0$. We have

$$H_1(c, p_s) = p_s(2) + c(1) - p_s(1) + 1 \le p_s(2) + c(1) + g(1) - p_s(1) + 1 = H_1(p_f, p_s) < 0,$$

which is a contradiction to the minimality of (p_f, p_s).

Case 2: $H_2(p_f, p_s) < 0$; this is symmetric to $H_1(p_s, p_f) < 0$. In this case we have

$$0 > H_2(p_f, p_s) = p_s(2) + p_f(1) - p_f(2) + 1 \ge p_s(2) - p_f(2) + 1 \ge p_s(2) - p_f(2) + g(2);$$

hence we have $p_s(2) + g(2) < p_f(2)$. If $g(1) < p_f(2) - p_s(1)$, then

$$||c + g||_\infty = ||p_f||_\infty \ge p_f(2) > ||p_s + g||_\infty,$$

which is a contradiction to PRIORITY(MAX) assigning g to c. Therefore, we have $g(1) \geq p_f(2) - p_s(1)$. Then, we have

$$\begin{aligned} H_1(c, p_s) &= p_s(2) + c(1) - p_s(1) + 1 = p_s(2) + p_f(1) - g(1) - p_s(1) + 1 \\ &\leq p_s(2) + p_f(1) - p_f(2) + 1 = H_2(p_f, p_s) < 0, \end{aligned}$$

which is also a contradiction.

◀

We are now ready to prove Lemma 17.

Proof of Lemma 17. Since $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ and $\alpha = 8/3$, we assume wlog that $p_f(2) > \alpha - 1 = 5/3$ and $p_s(1) > \alpha - 1 = 5/3$ (see Observation 10; the other case is symmetric). Then, we have

$$H_1(p_f, p_s) = p_s(2) + p_f(1) - p_s(1) + 1 \geq 0 \text{ by Lemma 18,}$$

which yields

$$p_s(2) + p_f(1) \geq 2/3.$$

Letting $p_f(1) = x$, we have

$$p_s(2) \geq 2/3 - x.$$

Note that

$$p_f \geq p'_f := (x, 5/3 + \epsilon); \quad \text{ and } \quad p_s \geq p'_s := (5/3 + \epsilon, 2/3 - x)$$

for sufficiently small $\epsilon > 0$. Thus, $(p'_s, p'_f) \notin \mathcal{F}$. Also notice $(p'_s, p'_f) \in \mathcal{L}(\alpha)$. Therefore, by Corollary 13, we have

$$V(p_f, p_s) \geq V(p'_f, p'_s) \geq 0.$$

By taking the limit $\epsilon \to 0$, we have

$$V(p_f, p_s) > \lim_{\epsilon \to 0} V(p'_f, p'_s) = \frac{(3x-1)^2}{3 \cdot (5 - 3x)} \geq 0,$$

which is a contradiction to $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ by Lemma 11. ◀

5 Balance Algorithm: Known OPT

In this section, we consider another priority-based greedy algorithm, PRIORITY(BAL). The rule BAL is defined as follows:

$$\text{BAL}(c, g) = \begin{cases} d(c) \cdot d(g) & c + g \leq \alpha \cdot \text{OPT} \\ \infty & \text{otherwise,} \end{cases} \tag{Bal-α}$$

where $d(v) := v(2) - v(1)$ measures the signed difference between v's right load and left load.

In other words, PRIORITY(BAL) tries to minimize the difference between the left and right loads over all machines, without violating a pre-defined threshold α. Note that this algorithm needs to know the value of OPT, which is wlog assumed to be 1 by scaling. The PRIORITY(BAL) algorithm keeps the machines in sorted order of the (signed) difference between the loads on the two coordinates (the left load minus the right load we say that the machines are maintained in left to right order where the rightmost (resp., leftmost) machine has the largest difference between the loads on the first and second coordinate (resp., second and first coordinate). Recall that a left (resp., right) vector is one whose first (resp., second) coordinate is larger. The PRIORITY(BAL) algorithm assigns a left (resp., right) vector to the rightmost (resp., leftmost) machine that can accommodate it, i.e., whose load on any dimension does not exceed the desired competitive ratio α after the assignment. In order to achieve it, given a left (right) vector the algorithm would prefer to assign to the most unbalanced right (left) machine that can accommodate the vector.

▶ **Theorem 19** (Upper Bound of Theorem 2). *$\textsc{Priority}(\textsc{Bal})$, knowing $\textsc{opt}$ a priori, is 2.25-competitive for the $2\textsc{DSched}$ problem.*

To prove Theorem 19, thanks to Lemma 14, it suffices to show the following.

▶ **Lemma 20.** *For $\alpha = 2.25$, we have $(\mathcal{R}_2(\textsc{Bal}) \cap \mathcal{L}(\alpha)) \setminus \mathcal{F} = \emptyset$.*

The remainder of this section is devoted to proving Lemma 20. Instead of analysing directly $\mathcal{R}_2(\textsc{Bal})$, we introduce a slightly modified rule of $\textsc{Bal}$, which is not subject to α:

$$\textsc{Bal-No-Lim}(c, g) = d(c) \cdot d(g) \tag{3}$$

Note that $\mathcal{R}_2(\textsc{Bal-No-Lim}) \cap \{v \mid v \in [0, \alpha]^2\} \subseteq \mathcal{R}_2(\textsc{Bal})$ and the subtle difference between $\mathcal{R}_2(\textsc{Bal-No-Lim})$ and $\mathcal{R}_2(\textsc{Bal})$. The closure $\mathcal{R}_2(\textsc{Bal-No-Lim})$ attempts to assign a vector g to only mitigate the difference of the left and right loads of the two machines. If we can assign g to the machine i^* that achieves this, then this assignment would be exactly the same as $\textsc{Priority}(\textsc{Bal})$ would make. However, in the closure $\mathcal{R}_2(\textsc{Bal-No-Lim})$, if g would overflow machine i^*, it doesn't add it to the other machine even if it would be possible under $\textsc{Priority}(\textsc{Bal})$. This is summarized in the following observation.

▶ **Observation 21.** *For two pairs of configuration $p = (c_a, c_b), p' = (c_a + g, c_b)$, such that p' is reachable in $\mathcal{R}_2(\textsc{Bal})$ by assigning a job of load $g \in [0, 1]^2$ into (a machine of load) c_a in the pair p, if $(c_a + g, c_b) \in \mathcal{R}_2(\textsc{Bal}) \setminus \mathcal{R}_2(\textsc{Bal-No-Lim})$ and $(c_a, c_b) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, then $c_b + g \nleq \vec{\alpha}$ and $(d(c_a) - d(c_b)) \cdot d(g) > 0$.*

For a pair of configuration (c_s, c_f), define

$$H_3(c_f, c_s) := d(c_s) - d(c_f) + 1 = c_s(2) - c_s(1) - c_f(2) + c_f(1) + 1$$

▶ **Lemma 22.** *For a pair of configuration $p = (p_s, p_f)$, if $H_3(p_f, p_s) < 0$ or $H_3(p_s, p_f) < 0$, then we have $p \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$.*

Proof. Assume for the purpose of contradiction that $p \in \mathcal{R}_2(\textsc{Bal-No-Lim})$. Assume $H_3(p_f, p_s) < 0$, since the case $H_3(p_s, p_f) < 0$ is symmetric. There must exist a vector g such that $p_f = c + g$, and $(c, p_s) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, for which $\textsc{Priority}$ $(\textsc{Bal-No-Lim})$ assigned to c, hence we have

$$\textsc{Bal-No-Lim}(c) - \textsc{Bal-No-Lim}(p_s) = (d(c) - d(p_s)) \cdot d(g) \leq 0.$$

We consider two cases.

Case 1. $g(2) > g(1)$: Clearly, $g(2) - g(1) \leq 1$. However, $d(p_s) - d(p_f) + 1 = H_3(p_f, p_s) < 0$, hence

$$d(p_s) < d(p_f) - 1 = d(c + g) - 1 = c(2) + g(2) - c(1) - g(1) - 1 \leq c(2) - c(1) = d(c).$$

Therefore, $(d(c) - d(p_s)) \cdot (g(2) - g(1)) > 0$, which is a contradiction.

Case 2. $g(2) \leq g(1)$: We have

$$c(2) - c(1) - 1 \geq c(2) + g(2) - c(1) - g(1) - 1 = p_f(2) - p_f(1) - 1 > p_s(2) - p_s(1).$$

Therefore, we have $H_3(c, p_s) < 0$ hence $(c, p_s) \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$, which is a contradiction. ◀

We now have all the pieces to prove Lemma 20.

Proof of Lemma 20. Assume for the purpose of contradiction that there exists a pair $p = (c_r, c_\ell) \in \mathcal{R}_2(\textsc{Bal}) \cap \mathcal{L}(\alpha)) \setminus \mathcal{F}$. Let $\langle p^\emptyset, p^1, \ldots p^n = p\rangle$ a sequence of reachable pairs i.e. for all i, $p^i \in \mathcal{R}_2(\textsc{Bal})$ and p^{i+1} is reachable from p^i by a single vector assignment under $\textsc{Priority}(\textsc{Bal})$.
Case 1. For all $i \in [n]$, $p^i \in \mathcal{R}_2(\textsc{Bal-No-Lim})$. Since $p = (c_r, c_\ell) \in \mathcal{R}_2(\textsc{Bal-No-Lim}) \cap \mathcal{L}(\alpha)$, assume wlog that $c_r(2) > \alpha - 1$, $c_\ell(1) > \alpha - 1$. In addition, since $(c_r, c_\ell) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, by Lemma 22, we have $H_3(c_r, c_\ell) = c_\ell(2) - c_\ell(1) - c_r(2) + c_r - 1 \geq 0$. Therefore, we have

$$0 \leq H_3(c_r, c_\ell) \leq H_3((c_r(1), \alpha - 1), (\alpha - 1, c_\ell(1)) = 3 - 2 \cdot \alpha + c_r(1) + c_\ell(2).$$

By setting $c_r(1) = x$, we have $c_\ell(2) \geq 2\cdot\alpha - 3 - x$. Note $x \in [0, 1]$ since $c_\ell(1) > 1$ and $(c_r, c_\ell) \notin \mathcal{F}$. For $\alpha = 2.25$, we have

$$V(c_r, c_\ell) > V((x, \alpha - 1), (\alpha - 1, 2 \cdot \alpha - 3 - x)) = \frac{(4x - 3)^2}{20 - 16x} \geq 0,$$

which is a contradiction to $p \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ by Lemma 11.

Case 2. $p^i \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$ for some $i \in [n]$. Let i be the first index such that $p^i \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$, let $p^{i-1} = (c_a, c_b)$, $p^i = (c_a + g, c_b)$. Note that $c_r \geq c_a$ and $c_\ell \geq c_b$. By Observation 21 we have $c_b + g \not\leq \vec{\alpha}$. Assuming wlog that $c_b(1) + g(1) > \alpha$, we have $c_b(1) > \alpha - g(1) > 1$ since $g \leq \vec{1}$ and $\alpha > 2$. For the same reason, we have $c_r(2) > \alpha - 1$, and $c_r(1), c_\ell(2) \leq 1$ (since $(c_r, c_\ell) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$).

Since $V(c_r, c_\ell)$ is monotone increasing, we have $V(c_r, c_\ell) > V(\langle c_a(1) + g(1), \alpha - 1\rangle, c_b)$. Moreover, by our assumption $p^{i-1} = (c_a, c_b) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, by Lemma 22, we have

$$\begin{aligned} H_3(c_a, c_b) &= c_b(2) - c_b(1) - c_a(2) + c_a(1) + 1 \quad \geq \quad 0, \\ c_a(1) &\geq \max\{c_a(2) - c_b(2) + c_b(1) - 1, 0\} \quad \geq \quad \max\{c_b(1) - c_b(2) - 1, 0\}. \end{aligned}$$

Recall that $c_r(1) \leq 1$, and we have $c_a(1) \geq c_b(1) - c_b(2) - 1$, and $g(1) > \alpha - c_b(1)$. Therefore, $1 \geq c_r(1) \geq g(1) + c_a(1) \geq \alpha - c_b(1) + c_a(1) \geq \alpha - c_b(2) - 1$, which yields $c_b(2) \geq \alpha - 2$. Thus,

$$V(c_r, c_\ell) > V(\langle \alpha - c_b(1) + \max\{c_b(1) - c_b(2) - 1, 0\}, \alpha - 1\rangle, \langle c_b(1), c_b(2)).$$

We lower bound V by considering two cases:
Case A. If $c_b(1) - c_b(2) < 1$, then

$$V(c_r, c_\ell) > V(\langle \alpha - c_b(1), \alpha - 1\rangle, \langle c_b(1), c_b(2)) \geq V(\langle \alpha - c_b(1), \alpha - 1\rangle, \langle c_b(1), c_b(1) - 1).$$

Case B. If $c_b(1) - c_b(2) \geq 1$, then

$$V(c_r, c_\ell) > V(\langle \alpha - c_b(2) - 1, \alpha - 1\rangle, \langle c_b(1), c_b(2)) \geq V(\langle \alpha - c_b(2), \alpha - 1\rangle, \langle 1 + c_b(2), c_b(2)).$$

Setting $x = c_b(1) - 1(\geq \alpha - 2)$ in the first case and $x = c_b(2)$ in the second case, we get that $x \geq \alpha - 2$ in both cases. So, we have

$$V(c_r, c_\ell) > V(\langle \alpha - 1 - x, \alpha - 1\rangle, \langle 1 + x, x\rangle) \geq 0.$$

By setting $\alpha = 2.25$, for $x \geq \alpha - 2 = 0.25$, we have

$$V(c_r, c_\ell) > V(\langle \alpha - 1 - x, \alpha - 1\rangle, \langle 1 + x, x\rangle) = \frac{(2x - 1)^2}{8x - 1} \geq 0,$$

which is a contradiction to $p \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ by Lemma 11. ◀

6 A Nearly Optimal Algorithm Against the Fractional Optimal Solution

Recall that all algorithms we developed and analyzed were based on the two most obvious lower bounds for the optimal solution, the total load vector of all jobs and the maximum job size on any dimension. Therefore, the benchmark we used can do better than the offline optimum solution. For example, consider three job vectors $(1, 1, 0), (1, 0, 1), (0, 1, 1)$ to be scheduled on 2 machines. Since one of the two machines must receive at least two jobs, the optimum makespan cannot be smaller than 2. However, this instance still has an average load of 1 on all dimensions and no job has size greater than 1 on any dimensions. In other words, the benchmark can distribute all jobs equally across all machines. For this reason, we will call this benchmark the fractional optimum solution.

▶ **Definition 23.** *For any number of dimensions $d \geq 1$, the optimum competitive ratio c^*_d against the fractional optimum solution is defined as*

$$\inf_A \sup_J \frac{\max_{i\in[m],k\in[d]} \Lambda_i^{J,m}(k)}{\max\{||\sum_{j\in J} v_j/m||_\infty, \max_{j\in J,k\in[d]} v_j(k)\}},$$

where A denotes an arbitrary deterministic online algorithm, J an arbitrary sequence of jobs, m the number of machines, and $\Lambda_i^{J,m}$ the load vector of machine i under the assignment of jobs J to machines $[m]$ by the algorithm A.

Our goal is to develop and analyze an algorithm that performs nearly as well as the fractional optimum solution.

▶ **Theorem 24.** *For any $d \geq 1$ and $\epsilon > 0$, assuming that the value of the optimum makespan*[5] *is known a priori, there exists a deterministic online algorithm for the vector scheduling problem whose competitive ratio is $(1 + \epsilon)c^*_d$. Further, the running time is polynomial in n for any fixed d, ϵ.*[6]

6.1 Assigning Jobs to Groups

The first stage of the algorithm is executed only when $m \geq (1 + \frac{1}{\epsilon})\alpha$ where $\alpha := \frac{250}{\epsilon^3} \log d$. We group machines so that every group has exactly α machines; to simplify the notation we assume that α is an integer to omit ceilings. The only one possible group that has less than α machines is discarded. We assign jobs to the (remaining) groups and obtain the following lemma using an algorithm and analysis very similar to [29]; hence, we defer the details to the full version of the paper.

▶ **Lemma 25.** *For a sufficiently small $\epsilon > 0$, there exists an online algorithm that assigns jobs to the groups, each consisting of $\alpha := \frac{250}{\epsilon^3} \log d$ machines, such that every group's total load is at most $(1 + \epsilon)\alpha d\vec{1}$.*

Note that the average load vector a machine should handle increases by a factor of at most $\frac{m}{m-(\alpha-1)} \leq \frac{1+1/\epsilon}{1+1/\epsilon-1} = 1 + \epsilon$. We also slightly modify each job's load vector: For each job j, we minimally increase v_j, so that we have $v_j(k) \geq (\epsilon/d)||v_j||_\infty$. This is wlog since increasing job load vectors can only increase the algorithm's makespan and we fixed the optimum to be 1.

▶ **Lemma 26.** *The preprocessing step increases the total load vector to at most $(1 + \epsilon)m\vec{1}$.*

[5] More accurately, the value of the fractional optimum makespan.

[6] More precisely, the running time is polynomial in n and $(d/\epsilon)^{d^{(d/\epsilon)^{O(d)}}}$.

Proof. For the sake of contradiction, suppose the total load is more than $(1+\epsilon)m$ on some dimension. It means the load increased by more than ϵm on the dimension. We know that job j contributes to the increase by at most $(\epsilon/d)||v_j||_\infty$. Thus, the increase is at most $\sum_j (\epsilon/d)||v_j||_\infty$. However, we know $\sum_j ||v_j||_\infty \leq md$ since the total load of all jobs across all dimensions is md. Therefore, the increase is at most ϵm, which is a contradiction. ◀

Since we are only concerned with assigning jobs to groups of machines at this stage, to simplify notation we pretend each group is a machine. By a machine i, we mean the i-th group which consists of α machines.

We now restate the problem: We are given $m' = \lfloor \frac{m}{\alpha} \rfloor$ machines. Let $[n]$ denote the set of all jobs arriving, which satisfies the following properties.

- Property (i): The total job load vector, i.e., $\sum_{j\in[n]} v_j$ is at most $m'\alpha(1+\epsilon)^2\vec{1}$.
- Property (ii): For all $j \in [n]$, $||v_j||_\infty \leq 1$.
- Property (iii): For all $j \in [n]$, $\min_k v_j(k) \geq (\epsilon/d)||v_j||_\infty$.

Our goal is to assign jobs to m' machines so that each group receives jobs of total load at most $(1+7\epsilon)\alpha\vec{1}$, which would immediately imply Lemma 25 by scaling ϵ.

The algorithm has two procedures. The algorithm pretends there are two sets M_1 and M_2 of machines, where $|M_1| = |M_2| = m'$. The first procedure assigns all jobs to machines M_1 and identifies a set J^2 of jobs, which will be assigned to machines M_2 by the second procedure. However, this is a shadow process: What really happens is that only jobs in $[n] \setminus J^2$ remain on machines M_1 and the other jobs J^2 are assigned to machines M_2. Further, the algorithm pairs machines between M_1 and M_2 arbitrarily and combine the load vectors of the paired machines. To prove Lemma 25, it suffices to show the following two lemmas (with scaling ϵ):

▶ **Lemma 27.** *The makespan of the assignment of $[n] \setminus J^2$ to M_1 is at most $((1+\epsilon)^5\alpha+1)\vec{1} \leq (1+6\epsilon)\alpha\vec{1}$.*

▶ **Lemma 28.** *The makespan of the assignment of J^2 to M_2 is at most $\epsilon\alpha\vec{1}$.*

We are now ready to describe the algorithm.

- **First procedure (assignment by a potential function)**: Let $\beta := (1+\epsilon)^3$. Let $f(x) := \beta^x$. Each job j is assigned to a machine $i \in M_1$ such that $\Phi(j)$ is minimized. For every $i \in M_1$, let $\Lambda^1_{i,j}$ denote machine i's load vector right after assigning job j to some machine in M_1. If $\Lambda^1_{i,j}(k) \geq \beta\alpha + 1$, then j is added to queue J^2 so that it can be scheduled by the second procedure.

$$\Phi_{i,k}(j) := f\left(\Lambda^1_{i,j}(k) - \frac{\beta}{m'}\sum_{j'\in[j]} v_{j'}(k)\right) \qquad \forall i \in M_1, j \in [n], k \in [d]$$

$$\Phi(j) := \sum_{i\in M_1}\sum_{k=1}^{d} \Phi_{i,k}(j)$$

- **Second procedure (assignment by greedy)**: This procedure is only concerned with the jobs J^2 that are passed from the first procedure. It allocates each job in J^2 (in the order that the jobs arrive in) to one of the machines in M_2 such that the resulting makespan, $\max_{i\in M_2,k\in[d]} \Lambda^2_{i,j}(k)$ is minimized; here $\Lambda^2_{i,j}$ is analogously defined as $\Lambda^1_{i,j}$ is defined in the first procedure.

Note that Lemma 27 immediately follows due to the way the first procedure is defined. The proof of Lemma 28 constitutes the heart of the analysis in this stage of the algorithm. Since this closely follows techniques in [29], we defer the details of this analysis to the full version of the paper.

6.2 Assigning Jobs to Machines Within Each Group

We need to define a fair amount of notation to formally describe our algorithm. We assume that the input consists of m machines and the average load of all jobs to be assigned is at most $(1+\epsilon)m$ on all dimensions for some $\epsilon > 0$.

For ease of reference, we list the following definitions.

- Let $\beta := \frac{\epsilon}{2md}$; $\Delta := \frac{\epsilon^2}{d(1+1/\beta)^d}$; and $\delta := \epsilon\beta\Delta/(2dm)$.
- A vector is said to be a type vector if it is in $\{0, \beta, 2\beta, \cdots, 1\}^d \setminus \{\vec{0}\}$ and has 1 on at least one dimension. Let $\mathcal{Q}$ denote the set of all type vectors. Note that $|\mathcal{Q}| \leq (1+1/\beta)^d \leq (2/\beta)^d$.
- We say a job j is small if $||v_j||_\infty < \Delta$; otherwise it is big.
- The volume of a job j is defined as its total size over all dimensions.

We discretize big jobs and small jobs in different manners:

- Big jobs: For a big job j, we round its load on every dimension down to the nearest integer multiple of δ. Let $\mathcal{B}$ denote the set of all possible load vectors of big jobs after discretization.
- Small jobs: For a small job j, let $p_j = ||v_j||_\infty$. Then, we discretize v_j/p_j by rounding each entry down to the nearest integer multiple of β. Let q_j denote the resulting discretized vector of v_j/p_j; note that $q_j \in \mathcal{Q}$. After rounding, we can express each small job j as $p_j q_j$.

We are now ready to describe our algorithm. Below, assume that jobs are already discretized. We will later show that the effect of discretization is negligible on the competitive ratio.

6.2.1 Building a decision tree

We build a decision tree T to assign big jobs. To simplify the analysis later, we assume wlog that the total load vector T receives is exactly $m(1+4\epsilon)\vec{1}$. Each node of the decision tree T corresponds to the current loads of all the m machines. Each node u has at most $m|\mathcal{B}|$ children. Each edge (u, w) is associated with a pair (i, j) where $j \in \mathcal{B}$ and $i \in [m]$, meaning that if a big job j is assigned to machine i, then the machine loads vectors change from u to w. Since the total volume of jobs is at most $(1+4\epsilon)md \leq 5md$ and every big job has volume at least Δ, the decision tree has depth at most $5md/\Delta$ and the number of nodes is at most $(m|\mathcal{B}|)^{5md/\Delta}$. We only need to keep nodes whose machine load vectors don't contradict the assumption that the total load of all jobs on each dimension is exactly $(1+4\epsilon)m$.

Given that every node of the decision tree T corresponds to a configuration (machine load vectors) that can be reached via a valid sequence of big jobs along with their assignment, our goal is to compute the minimum makespan we can achieve from each node u. Formally, if u is a leaf node, define $g(u)$ to be the makespan norm of the machine load vectors corresponding to u. Otherwise, let $u_{i,j}$ denote u's child such that the edge $(u, u_{i,j})$ is associated with (i, j). Then, define $g(u) := \min_i \max_j g(u_{i,j})$.

We can use the tree T to assign big jobs as follows. Let u be the node corresponding to the current machine load vectors. If a big job j arrives, then we assign j to machine i with the minimum $g(u_{i,j})$.

The following observation is immediate due to the optimal nature of the decision tree for big jobs. In other words, the observation says that the decision tree yields a nearly optimal algorithm against the fractional optimum.

▶ **Observation 29.** *Let r denote the root of T. Then, $g(r) \leq (1+4\epsilon)c_d^*$.*

Proof. Since we assumed that the total load vector is exactly $(1+4\epsilon)m\vec{1}$, the denominator in Definition 23 is exactly $(1+4\epsilon)$. Since we know the decision tree gives an optimal algorithm for big jobs, we have $g(r)/(1+4\epsilon) \leq c_d^*$, as desired. ◀

6.2.2 Batching smalls jobs of the same type

We will first describe how we batch small jobs and assign them using the above decision tree T assuming that we can wait until we collect enough volume of jobs for each type. For each type vector $q \in \mathcal{Q}$, we create a buffer $F(q)$. The buffer has capacity Δ/ϵ. When a small job j of vector $p_j q_j$ arrives, we add it to buffer $F(q_j)$; j uses p_j space of the buffer. There are two events that trigger emptying a buffer. When we empty a buffer $F(q)$, we encapsulate the load vectors of all jobs in $F(q)$ into a 'bin' vector $(\Delta/\epsilon)q$, and assign it using the decision tree T. The buffer is emptied when either we cannot add a job j since it would exceed the capacity Δ/ϵ or after all jobs arrive.

This procedure is well defined due to the following lemma.

▶ **Lemma 30.** *Every bin vector is in $\mathcal{B}$.*

Proof. Consider any bin vector $(\Delta/\epsilon)q$. To show that this is in $\mathcal{B}$, we need to show the following three: (i) it has size at least Δ on some dimensions; (ii) each of its entries is a multiple of δ; and (iii) it has size no more than 1 on every dimension. First, (i) follows since $||q||_\infty = 1$ due to the way we defined small job types. To see (ii), consider q's entry on each dimension – we know that its value must be $\ell\beta$ for some integer ℓ. So, it suffices to show that $(\Delta/\epsilon)(\ell\beta)/\delta$ is an integer. Recall that $\delta := \epsilon\beta\Delta/(2dm)$. Thus, we have, $(\Delta/\epsilon)(\ell\beta)/\delta = \frac{\Delta}{\epsilon}(\ell\beta)\frac{2dm}{\epsilon\beta\Delta}$, which is an integer assuming that $1/\epsilon$ is an integer. To see (iii), note that the maximum size over all dimensions is at most $(\Delta/\epsilon) = \frac{\epsilon}{d(1+1/\beta)^d} < 1$. ◀

6.2.3 Batching small jobs online

In the online setting we cannot wait to aggregate small jobs of the same type. To handle this issue, we pre-allocate one "bin" vector of each type. That is, before any jobs arrive, we pretend that one job of each load vector q arrives and assign it using the decision tree T. Then, batching jobs of the same type vector q in $F(q)$ is actually done on the machine that received the bin vector. Therefore, we can assign small jobs upon their arrival without waiting.

We have fully described our online algorithm to assign jobs upon their arrival. We now shift our focus to the analysis. When a job j is encapsulated into a type vector v of type $q \in \mathcal{Q}$, we say v contains job j.

▶ **Observation 31.** *Every bin vector of each type $q \in \mathcal{Q}$, possibly except one, has total size of jobs at least $(1-\epsilon)(\Delta/\epsilon)$.*

Proof. Since small jobs are aggregated only when they are of the same type, for each type $q \in \mathcal{Q}$, we can focus on the scalar quantities, job sizes p_j and the buffer size Δ/ϵ. The observation follows from the fact that we empty buffer $B(q)$ only when the total size of jobs in the buffer $B(q)$ exceeds $(1-\epsilon)(\Delta/\epsilon)$, or at the end after all jobs arrive. The only exception is due to the pre-allocation, which corresponds to emptying the buffer at the end. ◀

▶ **Lemma 32.** *If the total job load vector is at most $(1+\epsilon)m\vec{1}$, then the decision tree, due to batching and preallocation, receives jobs of total load vector at most $(1+4\epsilon)m\vec{1}$.*

Proof. By Observation 31, the total load vector T receives is at most $(1+\epsilon)m\vec{1}/(1-\epsilon) \leq (1+3\epsilon)m\vec{1}$, plus $\sum_{q\in\mathcal{Q}}(\Delta/\epsilon)q$. Further, we have $\sum_{q\in\mathcal{Q}}(\Delta/\epsilon)q = |\mathcal{Q}|(\Delta/\epsilon)\vec{1} \leq (1+1/\beta)^d \cdot \frac{\epsilon^2}{d(1+1/\beta)^d}\frac{1}{\epsilon}\vec{1} \leq \epsilon\vec{1}$. ◀

Therefore, the algorithm sends to the decision tree T big jobs (including bin vectors which are big) of total load vector at most $(1+4\epsilon)m\vec{1}$. Note that sending less loads only helps our algorithm. By

Observation 29, our algorithm's makespan is at most $(1+4\epsilon)c_d^*$-competitive if all jobs are discretized. We now show that when replacing each discretized vector with the original vector, every machine's load increases by at most $2\epsilon\vec{1}$. Knowing that the optimum makespan is 1, this will mean that the competitive ratio of our algorithm is at most $(1+6\epsilon)c_d^*$-competitive. By scaling ϵ appropriately, we obtain Theorem 24.

We complete the analysis by proving the following lemma.

▶ **Lemma 33.** *Restoring the discretized jobs load vectors to their original vectors increases each machine's load vector by at most $2\epsilon\vec{1}$.*

Proof. For the sake of contradiction, suppose the total load increases by more than 2ϵ on some fixed dimension d on some fixed machine i. In the first case, suppose at least ϵ increase was due to big jobs. Then, since discretizing a big job reduces its load by less than δ on each dimension, this means that the total number of big jobs assigned to the machine i is at least ϵ/δ. Since a big job has a volume Δ or more, the total volume of jobs assigned to machine i is at least $(\epsilon/\delta)\cdot\Delta = \epsilon/(\epsilon\Delta/2dm)\cdot\Delta = 2dm$, which is a contradiction to the fact that each machine has makespan at most $(1+4\epsilon)$, thus volume at most $(1+4\epsilon)d$. Now suppose at least ϵ increase was due to small jobs. Let S be the set of all small jobs assigned to machine i. We know that discretizing a small job j reduces its load by at most $p_j\beta$ on the fixed dimension d. Thus, we have $\sum_{j\in S} p_j\beta > \epsilon$. As a result, we have $\sum_{j\in S} p_j > \epsilon/\beta = \epsilon/(\epsilon/2md) = 2md$. This implies that the total volume of the jobs in S is at least $\sum_{j\in S} p_j \geq 2md$, which is a contradiction as before. ◀

6.3 Putting the Pieces Together

In Section 6.1 we showed if we use the first phase of the algorithm, then we can assign jobs to groups of machines so that each group has $m' = O(\frac{1}{\epsilon^3}\log d)$ machines and receives load at most $(1+\epsilon)m'\vec{1}$. Otherwise, we can pretend all jobs are assigned to the single group of all machines. In either case, we can assign jobs to groups of machines so that each group has $m' = O(\frac{1}{\epsilon^4}\log d)$ machines and receives load at most $(1+\epsilon)m'\vec{1}$. Then, using the procedure in 6.2, we can assign jobs to machines within group, so that each machine's load vector is at most $(1+6\epsilon)c_d^*\vec{1}$. Thus, we have found an online algorithm whose competitive ratio is $(1+\epsilon)c_d^*$ by appropriately scaling ϵ.

It now remains to show the running time of our algorithm. Since the running time is mostly dominated by the second phase, we will focus on the second phase. It is an easy exercise to see the running time is polynomially bounded by the size of decision tree and n. As discussed, the number of nodes is $(m|\mathcal{B}|)^{5md/\Delta}$, where $|\mathcal{B}| \leq (2/\delta)^d$. By the above discussion, we have $m = O(\frac{1}{\epsilon^4}\log d)$. Recall that $\beta := \frac{\epsilon}{2md}$, $\Delta := \frac{\epsilon^2}{d(1+1/\beta)^d}$, $\delta := \epsilon\beta\Delta/(2dm)$. By calculation, one can show that the tree size is $(d/\epsilon)^{d^{(d/\epsilon)^{O(d)}}}$. Thus, we have shown the running time.

This completes the proof of Theorem 24.

7 Open Problems

This paper gives the first non-trivial results for the online vector scheduling problem with a small number of dimensions. The most interesting open question is to better understand the competitive ratio of "practical" algorithms when $d > 2$. For instance, what is the competitive ratio of PRIORITY(MAX) when $d > 2$? Or, can we extend PRIORITY(BAL) for $d = 2$ to higher dimensions? Even for $d = 2$, in our analysis, we used as the lower bound the fractional optimum where job vectors can be fractionally assigned to machines. This is inherently limited by the integrality gap of the fractional assignment. Can we obtain better lower bounds for the true optimum, and thereby improve the competitive ratio for the problem? For instance, for $d = 2$, is there a 2-competitive algorithm?

References

1. Susanne Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999.
2. Susanne Albers. On randomized online scheduling. In *STOC*, pages 134–143, 2002.
3. Susanne Albers. Online algorithms: a survey. *Math. Program.*, 97(1-2):3–26, 2003.
4. Susanne Albers and Matthias Hellwig. Semi-online scheduling revisited. *Theor. Comput. Sci.*, 443:1–9, 2012.
5. Yossi Azar. On-line load balancing. In *Online Algorithms, The State of the Art.*, pages 178–195, 1996.
6. Yossi Azar, Ilan Reuven Cohen, Amos Fiat, and Alan Roytman. Packing small vectors. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1511–1525, 2016.
7. Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and F. Bruce Shepherd. Tight bounds for online vector bin packing. In *Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, pages 961–970, 2013.
8. Yossi Azar, Ilan Reuven Cohen, and Debmalya Panigrahi. Randomized algorithms for online vector load balancing. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 980–991, 2018.
9. Yossi Azar and Oded Regev. On-line bin-stretching. *Theor. Comput. Sci.*, 268(1):17–41, 2001.
10. Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.
11. Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.
12. Martin Böhm, Jiří Sgall, Rob van Stee, and Pavel Veselý. A two-phase algorithm for bin stretching with stretching factor 1.5. *J. Comb. Optim.*, 34(3):810–828, 2017.
13. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, USA, 1998.
14. Allan Borodin, Morten N Nielsen, and Charles Rackoff. (incremental) priority algorithms. *Algorithmica*, 37(4):295–326, 2003.
15. Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004.
16. Edward Grady Coffman Jr, Michael Randolph Garey, and David Stifler Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.
17. Richard Cole, Vasilis Gkatzelis, and Gagan Goel. Mechanism design for fair division: allocating divisible items without payments. In *ACM Conference on Electronic Commerce, EC '13, Philadelphia, PA, USA, June 16-20, 2013*, pages 251–268, 2013.
18. Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta Cybern.*, 9(2):107–119, 1989.
19. Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000*, pages 202–210, 2000.
20. Michaël Gabay, Nadia Brauner, and Vladimir Kotov. Improved lower bounds for the online bin stretching problem. *4OR*, 15(2):183–199, 2017.
21. Michaël Gabay, Vladimir Kotov, and Nadia Brauner. Online bin stretching with bunch techniques. *Theor. Comput. Sci.*, 602:103–113, 2015.
22. Gábor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. *SIAM J. Comput.*, 22(2):349–355, 1993.
23. M. R. Garey, Ronald L. Graham, David S. Johnson, and Andrew C. Yao. Resource constrained scheduling as generalized bin packing. *J. Comb. Theory, Ser. A*, 21(3):257–298, 1976.
24. Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

25 Todd Gormley, Nick Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000, January 9-11, 2000, San Francisco, CA, USA.*, pages 564–565, 2000.

26 R. L. Graham. Bounds for certain multiprocessing anomalies. *Siam Journal on Applied Mathematics*, 1966.

27 R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

28 J. F. Rudin III. Improved bound for the on-line scheduling problem. *PhD thesis, The University of Texas at Dallas*, 2001.

29 Sungjin Im, Nathaniel Kell, Janardhan Kulkarni, and Debmalya Panigrahi. Tight bounds for online vector scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 525–544, 2015.

30 Sungjin Im, Nathaniel Kell, Debmalya Panigrahi, and Maryam Shadloo. Online load balancing on related machines. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 30–43, 2018.

31 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *Proc. 46th ACM Symposium. on Theory of Computing (STOC)*, pages 313–322, 2014.

32 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive flow time algorithms for polyhedral scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 506–524, 2015.

33 David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

34 David S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, 1974.

35 David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing*, 3(4):299–325, 1974.

36 David S. Johnson, Alan J. Demers, Jeffrey D. Ullman, M. R. Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3(4):299–325, 1974.

37 David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *J. Algorithms*, 20(2):400–430, 1996.

38 Hans Kellerer and Vladimir Kotov. An efficient algorithm for bin stretching. *Oper. Res. Lett.*, 41(4):343–346, 2013.

39 Hans Kellerer, Vladimir Kotov, and Michaël Gabay. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *J. Scheduling*, 18(6):623–630, 2015.

40 Hans Kellerer, Vladimir Kotov, Maria Grazia Speranza, and Zsolt Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21(5):235–242, 1997.

41 Gunho Lee, Byung-Gon Chun, and Randy H Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, volume 11, 2011.

42 Elisabeth Lübbecke, Olaf Maurer, Nicole Megow, and Andreas Wiese. A new approach to online scheduling: Approximating the optimal competitive ratio. *ACM Trans. Algorithms*, 13(1):15:1–15:34, 2016. URL: `https://doi.org/10.1145/2996800`, `doi:10.1145/2996800`.

43 Adam Meyerson, Alan Roytman, and Brian Tagiku. Online multidimensional load balancing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013.*, pages 287–302, 2013.

44 Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. Faircloud: sharing the network in cloud computing. In *ACM SIGCOMM*, pages 187–198. ACM, 2012.

45 Kirk Pruhs, Jiří Sgall, and Eric Torng. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis*, pages 15–1, 2004.
46 Jiří Sgall. On-line scheduling. In *Online Algorithms*, pages 196–231, 1996.
47 Jiří Sgall. Online scheduling. In *Algorithms for Optimization with Incomplete Information, 16.-21. January 2005*, 2005.
48 Jiří Sgall. Online bin packing: Old algorithms and new results. In *CiE*, volume 8493 of *Lecture Notes in Computer Science*, pages 362–372. Springer, 2014.