Online Load Balancing on Related Machines

Sungjin Im
University of California at Merced
Merced, CA, USA
sim3@ucmerced.edu

Nathaniel Kell
Duke University
Durham, NC, USA
kell@cs.duke.edu

Debmalya Panigrahi
Duke University
Durham, NC, USA
debmalya@cs.duke.edu

Maryam Shadloo
University of California at Merced
Merced, CA, USA
mshadloo@ucmerced.edu

ABSTRACT

We give a constant-competitive algorithm for the problem of assigning jobs online to machines with non-uniform speed (called related machines) so as to optimize any ℓ_q-norm of the machine loads. Previously, such a result was only known for the special case of the makespan, or ℓ_∞. norm. We also extend these results to obtain tight bounds for the problem of vector scheduling on related machines, where no results were previously known even for the makespan norm. To obtain our results, we employ a convex relaxation of the ℓ_q-norm objective and use a *continuous greedy* algorithm to solve this convex program online. To round the fractional solution, we then use a novel restructuring of the instance that we call *machine smoothing*. This is a generic tool that reduces a problem on related machines to a set of problem instances on identical machines, and we hope it will be useful in other settings with non-uniform machine speeds as well.

CCS CONCEPTS

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

KEYWORDS

online algorithms, scheduling, load balancing

ACM Reference Format:
Sungjin Im, Nathaniel Kell, Debmalya Panigrahi, and Maryam Shadloo. 2018. Online Load Balancing on Related Machines. In *Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC'18).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3188745.3188966

1 INTRODUCTION

In this paper, we consider the online *load balancing* problem of assigning jobs to machines with non-uniform speeds (called *related machines*). The online load balancing problem has a long history for optimizing the *makespan* norm (maximum machine load or ℓ_∞ norm of machine loads) on machines of uniform speed (identical machines) [1, 11, 12, 20, 21, 23, 26, 30]. This problem has also been studied for the more general setting of unrelated machines, where the processing time of a job on a machine is arbitrary, and a tight competitive ratio of $O(\log m)$ is known for the problem (where m is the number of machines) [3, 10, 14]. In many situations, however, other ℓ_q-norms of machine loads are more relevant than the makespan norm: e.g., the 2-norm is suitable for disk storage [16, 18], whereas q between 2 and 3 is used for modeling energy consumption [2, 33, 37]. This has led to the design of load balancing algorithms for optimizing arbitrary ℓ_q-norms (or simply q-norms) of machine loads, and tight competitive ratios of $O(1)$ for identical machines [4] and $O(q)$ for unrelated machines [5, 14, 15] are known. However, for related machines, the only previous result was a constant-competitive algorithm for the makespan norm, using the so-called *slowest fit* algorithm [3, 13]. No results were known for any other q-norm prior to our work. Our first result is to show *the first constant-competitive algorithms for optimizing arbitrary q-norms on related machines.*

Recent literature has further expanded the scope of the job scheduling problem to vector jobs that have multiple dimensions, the resulting problem being called *vector scheduling* [8, 17, 29, 32]. This problem arises in scheduling on 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, network bandwidth, and storage [19, 22, 28, 29, 31, 34]. Recently, Im *et al.* [27] showed that for vector scheduling with the makespan norm, competitive ratios of $O(\log d/\log\log d)$ and $O(\log d + \log m)$ are tight for identical and unrelated machines respectively, where d is the number of dimensions and m is the number of machines. They also extended these results to arbitrary ℓ_q-norms. In many data center applications, the situation is between these two extremes of identical and unrelated machines, and resembles the related machines scenario. In other words, machines have non-uniform speeds and the load created by a vector job on any dimension of a machine is inversely proportional to the machine speed. But, no results were known prior to our work for vector scheduling on related machines, either for the makespan norm or for arbitrary ℓ_q norms.

There are two natural variants of vector scheduling depending on whether machine speed is identical across dimensions (we call this *homogeneous* speeds) or different across dimensions (we call this *heterogeneous* speeds). We show a sharp contrast between these two settings: *we give algorithms minimizing makespan and ℓ_q norms in the homogeneous setting that match those for identical machines,*

STOC'18, June 25–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5559-9/18/06…$15.00
https://doi.org/10.1145/3188745.3188966

and show lower bounds in the heterogeneous setting that match those for unrelated machines. The lower bounds for vector scheduling on related machines with heterogeneous speeds are deferred to the full version of the paper because of space constraints.

Preliminaries and Results: First, we set up some standard notation. In online scheduling, a set of n jobs arrive online and each job must be irrevocably assigned to one of m machines immediately on arrival. Each job j has a non-negative *size* p_j. In vector scheduling, p_j is a vector of d dimensions, $p_j = \langle p_j(1), p_j(2), \ldots, p_j(d)\rangle$. Each machine i has a non-negative speed s_i that is given offline. In vector scheduling, s_i is a vector $\langle s_i(1), s_i(2), \ldots, s_i(d)\rangle$, where $s_i(1) = s_i(2) = \ldots = s_i(d)$ (denoted s_i) in the homogeneous setting. When job j is assigned to machine i, it produces a *load* of p_j/s_i. In vector scheduling, the load is $p_j(k)/s_i(k) = p_{ij}(k)$ in dimension k. The load produced by a set of jobs is the sum of their individual loads. The *load vector* is denoted $\Lambda = \langle \Lambda_1, \Lambda_2, \ldots, \Lambda_m\rangle$, where Λ_i is the total load on machine i. For vector scheduling, every dimension k has its own load vector, denoted $\Lambda(k) = \langle \Lambda_1(k), \Lambda_2(k), \ldots, \Lambda_m(k)\rangle$, where $\Lambda_i(k)$ is the total load on machine i in dimension k.

In vector scheduling, the makespan objective is given by:

$$\max_{k=1}^{d} ||\Lambda(k)||_\infty = \max_{k=1}^{d} \max_{i=1}^{m} \Lambda_i(k).$$

For the problem of minimizing makespan in vector scheduling, we show the following result.

Theorem 1. *For online vector scheduling on related machines for minimizing makespan:*

(1) *(Section 3) For homogeneous speeds, we give a deterministic $O(\log d/\log\log d)$-competitive algorithm. This is asymptotically tight since it matches a known lower bound for identical machines [27].*
(2) *(Deferred to full version) For heterogeneous speeds, we give a lower bound of $\Omega(\log d + \log m)$ on the competitive ratio. This is asymptotically tight since it matches a known upper bound for unrelated machines [8, 27, 32].*

Now we state our results for optimizing arbitrary ℓ_q-norms. First, we consider the scalar scheduling problem. The ℓ_q-norm objective is given by:

$$||\Lambda||_q = \left(\sum_{i=1}^{m}(\Lambda_i)^q\right)^{1/q}$$

We obtain the following result.

Theorem 2. *For online (scalar) scheduling on related machine for minimizing ℓ_q-norms (Section 4), we give a deterministic algorithm with a constant competitive ratio. This is asymptotically tight because online scheduling has a constant lower bound even for identical machines [1, 12, 20, 23, 26].*

Next, we consider optimizing ℓ_q-norms in vector scheduling. Our objective is given by:

$$\max_{k=1}^{d} ||\Lambda(k)||_q = \max_{k=1}^{d} \left(\sum_{i=1}^{m}(\Lambda_i(k))^q\right)^{1/q}$$

We obtain the following result.

Theorem 3. *For online vector scheduling on related machines for minimizing ℓ_q-norms:*

(1) *(Section 5) For homogeneous speeds, we give a deterministic algorithm with a competitive ratio of $O(\log^c d)$ for some constant c. This is tight up to the value of the constant c, by a known lower bound for identical machines [27].*
(2) *(Deferred to full version) For heterogeneous speeds, we give a lower bound of $\Omega(\log d + q)$ on the competitive ratio. This is asymptotically tight since it matches a known upper bound for unrelated machines [27].*

Note that Theorem 2 follows as a corollary of Theorem 3. However, our vector scheduling algorithm uses our scalar scheduling algorithm as a subroutine; consequently, the proof of Theorem 3 relies on an independent proof of Theorem 2. Therefore, we present our scalar scheduling results before presenting our vector scheduling results for arbitrary q-norms. We note that homogeneous related machines do not admit algorithms that optimize all norms *simultaneously* unlike identical machines. Thus we can only have algorithms that optimize each specific q-norm.

Techniques: A key tool in our algorithms is what we call *machine smoothing*. Imagine grouping together machines with similar speeds. Then, one can employ a two-stage algorithm that assigns each job to a machine group, and then use an identical machines algorithm within each machine group. Unfortunately, this simple grouping does not make the assignment problem easier as the number of machines in each group might be completely arbitrary. It turns out that the assignment of jobs to groups is facilitated if we can ensure that the cumulative *processing power* in a group exponentially increases as we move to slower groups. (The cumulative processing power for the makespan objective is simply the sum of speeds of machines in the group; for other ℓ_q-norms, this definition is suitably generalized.) So, now we have two objectives: group machines with similar speeds, but also ensure exponentially increasing processing powers of the groups in decreasing speed order. We show that we can transform any given instance into an instance that satisfies these goals simultaneously, which we call a *smoothed instance*, while only sacrificing a constant factor in the competitive ratio of the algorithm.

It turns out that the machine smoothing technique is essentially sufficient for solving the makespan minimization problem in vector scheduling, since the assignment of jobs to machine groups in a smoothed instance can be done by simulating the slowest-fit strategy used for scalar scheduling. However, for other ℓ_q-norms, even for scalar scheduling, we need to work harder in designing the algorithm to assign jobs to machine groups in a smoothed instance. In particular, we use a two-step approach. First, we use a *continuous greedy* algorithm on a suitably chosen fractional relaxation of the norm to produce a competitive fractional solution. Next, we use an online rounding algorithm to produce an integer assignment from the fractional solution. In the case of vector scheduling for arbitrary ℓ_q-norms, an additional complication is caused by the fact that the continuous greedy algorithm can produce unbalanced loads on different dimensions since it follows the gradient for a single objective, thereby leading to a large competitive ratio. To avoid this difficulty, we use the assignment produced by the continuous greedy algorithm only as an *advice* on the approximate speed of the machine group that a fractional job should be assigned to. We then use a different algorithm to make the actual assignment of the fractional job to a machine group similar to the advice, but not necessarily to the exact same group.

Related Work: In the interest of space, we will only state a small subset of related results and refer the reader to more detailed surveys [6, 35, 36] for other results. The online job scheduling problem was introduced by Graham [24], who showed that list scheduling has a competitive ratio of $(2 - 1/m)$ for the makespan objective on identical machines. Currently, the best known upper bound is 1.9201 [1, 11, 21, 30], while the best lowerbound is 1.880 [1, 12, 20, 23, 26]. For the related machines setting, the slowest-fit algorithm is 2-competitive [13] (given the optimal makespan a priori), but for unrelated machines, the optimal competitive ratio is $\Theta(\log m)$ [3, 10]. This problem was generalized to arbitrary q-norms by [4] for identical machines and by [5, 14] for unrelated machines. The only previous result for related machines was the competitive ratio of 2 achieved by the slowest-fit algorithm for the makespan norm [13].

For vector scheduling for identical machines, a PTAS is known for any fixed d [17]. For unrelated machines, the best known approximation factor is $O(\log d/\log\log d)$ [25] and a constant lower bound is known [17]. In the online setting, $O(\log d)$-competitive algorithms were given for identical machines [7, 32]. A recent work [27] improved these results by giving tight bounds of $O(\log d/\log\log d)$ for identical machines and $O(\log d + \log m)$ for unrelated machines. They also extended these results to arbitrary q-norms, giving tight bounds of $O((\frac{\log d}{\log\log d})^{\frac{q-1}{q}})$ and $O(\log d + q)$ for identical and unrelated machines.

2 MACHINE SMOOTHING

One of the main ideas that we use throughout our algorithms is that of *machine smoothing*. There are two properties that we wish to derive from machine smoothing: that machines in a single group have the same speed and that a slower group has processing power at least as much the sum over all its faster groups. To ensure both properties simultaneously, simply grouping the given machines is not sufficient – instead, we need to modify machine speeds in the given instance and create new machines. The goal of this section is to show that such modification is valid, i.e., there is a mapping between assignments made on the original instance and the smoothed instance that only changes the objective by a constant factor.

We use a parameter $\gamma := q/(q-1)$ for defining smoothed instances for the q-norm. For the makespan norm, we define $\gamma := 1$. Using this parameter, we define the processing power of a group, (denoted $S(G)$ for group G) $S(G) := \sum_{i\in G} s_i^\gamma$; note that for the makespan norm, the processing power is simply the sum of speeds of the machines in the group, i.e., $S(G) := \sum_{i\in G} s_i$.

Definition 1. An instance is *smoothed* if the machines are partitioned into groups $G_0, G_1, \ldots$ such that:

- **Property 1**: All machines in a group have *equal* speed.
- **Property 2**: $S(G_\ell) \geq S(G_0) + S(G_1) + \cdots + S(G_{\ell-1})$ for all groups G_ℓ, where $S(G) := \sum_{i\in G} s_i^\gamma$.
- **Property 3**: For any ℓ, the value of s_i^γ for machines $i \in G_\ell$ is at least twice that of machines $i \in G_{\ell+1}$. For the makespan norm, this means that machines in G_ℓ are at least twice as fast as those in $G_{\ell+1}$.

Machine Smoothing for Makespan Norm. First, we give an algorithm that converts an arbitrary set of machines M to a smoothed set of machines M' for the makespan norm (see Fig. 1). By scaling and rounding all machine speeds to powers of 2, we assume that all machine speeds are 2^{-t} for some integer $t \geq 0$ with the fastest machine having a speed of 1. Clearly, this changes the makespan by at most a factor of 2.

First, we order the machines in M in non-increasing order of speed, breaking ties arbitrarily. Let us overload notation and call this sorted order M as well. The first group G_0 comprises only the first machine in M, i.e., a machine with speed 1. To define a group G_ℓ for $\ell \geq 1$, we discard the prefix of M comprising machines in $G_0 \cup G_1 \cup \ldots \cup G_{\ell-1}$ and from the remaining machines in M, choose a minimal prefix such that their total processing power (sum of speeds) is exactly 2^ℓ. Note that if the total processing power of the remaining machines is at least 2^ℓ, this is always possible since machine speeds are (non-positive) powers of 2. If the total processing power of the remaining machines is less than 2^ℓ, we simply discard these remaining machines (call these discarded machines G_{L+1} where L is the index of the last group of machines). Thus, for each group G_ℓ with $0 \leq \ell \leq L$, we have $S(G_\ell) = 2^\ell$.

Let $s_{\min}(G_\ell)$ denote the lowest speed among all machines in G_ℓ. To create our smoothed set of machines M', for each ℓ we replace G_ℓ with a new set of machines G'_ℓ whose speeds are all equal to $s_{\min}(G_\ell)$. The number of machines in G'_ℓ is chosen such that the processing power is preserved, i.e., $S(G'_\ell) = S(G_\ell)$. Note that G'_ℓ may have more machines than G_ℓ. Finally, we combine all groups in $G'_0, G'_1, \ldots$ that have the same speed to form a new set of groups $G''_0, G''_1, \ldots$, which form a smoothed instance.

The next lemma claims that any assignment in the original instance can be mapped to an assignment in the smoothed instance, and vice-versa, such that the makespan only changes by a constant.

Lemma 4. *For any set M of machines (with homogeneous speeds in the case of vector scheduling), let M' be the set of smoothed machines defined by the smoothing procedure for the makespan norm. Then there exist two mappings $g : M \to M'$ and $g' : M' \to M$ such that if we are given a schedule on machines in M (resp., M'), then scheduling jobs assigned to machine i on machine $g(i)$ (resp., $g'(i)$) produces a schedule with makespan at most a constant factor more than that of the original schedule on M (resp., M').*

Proof. To define our mappings, we will use both sets of groups $G_0, G_1, \ldots$ on machines M and $G'_0, G'_1, \ldots$ on machines M'. (Note that $\{G''_\ell\}_\ell$ and $\{G'_\ell\}_\ell$ are groups defined on the same set of machines M'; so it is sufficient to output an assignment on $G'_0, G'_1, \ldots$.)

First, we define the mapping $g(\cdot)$ (see the bottom of Fig. 1 for an illustration). We partition the machines in group $G_{\ell+1}$ for $0 \leq \ell \leq L$ into sets $\{H_{i'}\}_{i' \in G'_\ell}$, where all machines in $H_{i'}$ are mapped to a single machine $i' \in G'_\ell$. Since no machine in $G_{\ell+1}$ has a speed greater than $s_{i'}$ and $S(G_{\ell+1}) \leq 2S(G'_\ell)$, we can ensure that $S(H_{i'}) \leq 3s_{i'}$, e.g., by using a greedy partitioning algorithm. Thus, this mapping increases the makespan of a schedule by at most a factor of 3. Additionally, we map the unique machine in G_0 to itself, leading to an overall increase in the makespan by at most a factor of 4.

Now, we define the mapping $g'(\cdot)$ from machines M' to M (see the top of Fig. 1 for an illustration). We partition the machines in

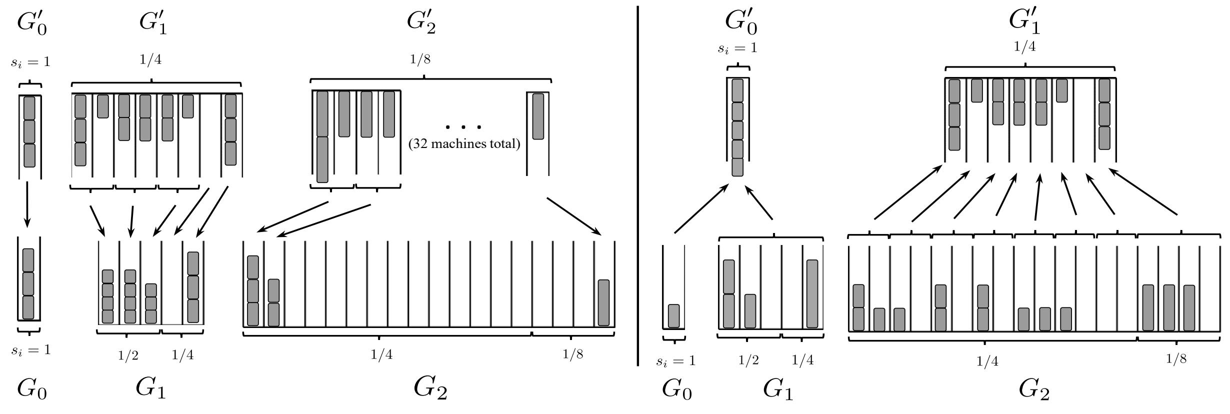

Figure 1: Illustration of machine smoothing. Here our original instance of machines M consists of 24 machines, 1 machine of speed 1, 3 machines of speed 1/2, 16 machines of speed 1/4, and 4 machines of speed 1/8. These machines are then grouped into G_0, G_1, and G_2 as shown. Our smoothed instance of machines M' consists of groups G'_0 (1 machine of speed 1), G'_1 (8 machines of speed 1/4) and G'_2 (32 machines of speed 1/8). The left and right illustrations demonstrate mappings $g'(\cdot)$ and $g(\cdot)$, respectively.

group G'_ℓ for $0 \le \ell \le L$ into sets $\{H_i\}_{i \in G_\ell}$, where all machines in H_i are mapped to a single machine $i \in G_\ell$. Since every machine in G_ℓ has at least the speed of machines in G'_ℓ, $S(G_\ell) = S(G'_\ell)$, and all speeds are powers of 2, we can ensure that $S(H_i) = s_i$, e.g., by again using a greedy partitioning algorithm. Thus, this mapping does not increase the makespan of a schedule. □

Machine Smoothing for q-norms. For an arbitrary q-norm, we need to modify our machine smoothing procedure because the definition of processing power of a group, and consequently that of a smoothed instance, is different from that for makespan. Thus in this section, we give an algorithm for converting machines M to a set of smoothed machines M' when optimizing for arbitrary q-norms where $q \ge 2$ (since the case of $q = 1$ is trivial).

Recall that in the definition of smoothed instance for q-norms, we define total processing power of a group $S(G_\ell) = \sum_{i \in G_\ell} s_i^\gamma$, where $\gamma = q/(q-1)$. Define $\Gamma := 2^{1/\gamma}$. Again by scaling and rounding, we can assume that the fastest machine has speed exactly 1 and that all machines in M have speeds that are (non-positive) powers Γ, i.e., $1, \Gamma^{-1}, \Gamma^{-2}, \ldots$. This changes the q-norm by at most a factor of γ which is at most 2 since $q \ge 2$.

We again we order the machines in M in non-increasing order of speed, breaking ties arbitrarily. Let us overload notation and call this sorted order M as well. The first group G_0 comprises only the first machine in M, i.e., a machine with speed 1. To define a group G_ℓ for $\ell \ge 1$, we discard the prefix of M comprising machines in $G_0 \cup G_1 \cup \ldots \cup G_{\ell-1}$ and from the remaining machines in M, choose a minimal prefix such that their total processing power (sum of s_i^γ) is exactly 2^ℓ. Note that if the total processing power of the remaining machines is at least 2^ℓ, this is always possible since machine speeds are (non-positive) powers of Γ (and thus each s_i^γ is a non-positive power of 2). If the total processing power of the remaining machines is less than 2^ℓ, we simply discard these remaining machines (call these discarded machines G_{L+1} where L is the index of the last group of machines). Thus, for each group G_ℓ with $0 \le \ell \le L$, we have $S(G_\ell) = 2^\ell$.

Let $s_{\min}(G_\ell)$ denote the lowest speed among all machines in G_ℓ. To create our smoothed set of machines M', for each ℓ we replace G_ℓ with a new set of machines G'_ℓ whose speeds are all equal to $s_{\min}(G_\ell)$. The number of machines in G'_ℓ is chosen such that the processing power is preserved, i.e., $S(G'_\ell) = S(G_\ell)$. Note that G'_ℓ may have more machines than G_ℓ. Finally, we combine all groups in $G'_0, G'_1, \ldots$ that have the same speed to form a new set of groups $G''_0, G''_1, \ldots$, which form a smoothed instance.

We now generalize Lemma 4 to arbitrary q-norms.

Lemma 5. *For any set M of machines (with homogeneous speeds in the case of vector scheduling), let M' be the set of smoothed machines defined by the smoothing procedure for the q-norm. Then there exist two mappings $g : M \to M'$ and $g' : M' \to M$ such that if we are given a schedule on machines in M (resp., M'), then scheduling jobs assigned to machine i on machine $g(i)$ (resp., $g'(i)$) produces a schedule with q-norm at most a constant factor more than that of the original schedule on M (resp., M').*

Proof. To define our mappings, we will use both sets of groups $G_0, G_1, \ldots$ on machines M and $G'_0, G'_1, \ldots$ on machines M'. (Note that $\{G''_\ell\}_\ell$ and $\{G'_\ell\}_\ell$ are groups defined on the same set of machines M'; so it is sufficient to output an assignment on $G'_0, G'_1, \ldots$.)

First, we define the mapping $g(\cdot)$. We partition the machines in group $G_{\ell+1}$ for $0 \le \ell \le L$ into sets $\{H_{i'}\}_{i' \in G'_\ell}$, where all machines in $H_{i'}$ are mapped to a single machine $i' \in G'_\ell$. Since no machine in $G_{\ell+1}$ has a speed greater than $s_{i'}$ and $S(G_{\ell+1}) \le 2S(G'_\ell)$, we can ensure that $S(H_{i'}) \le 3s_{i'}^\gamma$, e.g., by using a greedy partitioning algorithm. We now want to show that the q^q-norm of the loads on machines in $H_{i'}$ only increases by a factor of at most 3^q when the loads are relocated to machine i'. Let u_t denote the total volume machine $t \in H_{i'}$ (i.e, sum of p_j on machine t). Then, we have,

$$\sum_{t \in H_{i'}} \left(\frac{u_t}{s_t}\right)^q = S(H_{i'}) \sum_{t \in H_{i'}} \frac{s_t^\gamma}{S(H_{i'})} \left(\frac{u_t}{s_t^\gamma}\right)^q$$
$$\geq S(H_{i'}) \left(\sum_{t \in H_{i'}} \frac{s_t^\gamma}{S(H_{i'})} \cdot \frac{u_t}{s_t^\gamma} \right)^q$$
$$= \frac{s_{i'}^{\gamma(q-1)}}{S(H_{i'})^{q-1}} \left(\frac{\sum_{t \in H_{i'}} u_t}{s_{i'}}\right)^q \geq \frac{1}{3^q} \left(\frac{\sum_{t \in H_{i'}} u_t}{s_{i'}}\right)^q,$$

where the first inequality follows from the convexity of x^q; the second inequality follows since $S(H_{i'})/s_{i'}^\gamma \leq 3$. Finally, note that the first group G'_0 processes jobs relocated not only from G_1 but also from G_0. Hence the q^q-norm increases by a factor of at most 6^q, meaning that the optimal q-norm increases by at most a constant factor.

Now, we define the mapping $g'(\cdot)$ from machines M' to M. We partition the machines in group G'_ℓ for $0 \leq \ell \leq L$ into sets $\{H_i\}_{i \in G_\ell}$, where all machines in H_i are mapped to a single machine $i \in G_\ell$. Since every machine in G_ℓ has at least the speed of machines in G'_ℓ, $S(G_\ell) = S(G'_\ell)$, and all s_i^γ are powers of 2, we can ensure that $S(H_i) = s_i^\gamma$, e.g., by again using a greedy partitioning algorithm. Observe that $|H_i| = s_i^\gamma/(s'_\ell)^\gamma$, where s'_ℓ denotes the uniform speed of machines in G'_ℓ. We again let u_t be the volume of jobs assigned on machine $t \in H_i$. Thus we we have

$$\sum_{t \in H_i} \left(\frac{u_t}{s'_\ell}\right)^q \geq |H_i| \cdot \left(\frac{\sum_{t \in H_i} u_t}{s'_\ell |H_i|}\right)^q$$
$$= \left(\frac{1}{|H_i|}\right)^{q-1} \cdot \left(\frac{s_i}{s'_\ell}\right)^q \cdot \left(\frac{\sum_{t \in H_i} u_t}{s_i}\right)^q = \left(\frac{\sum_{t \in H_i} u_t}{s_i}\right)^q,$$

where the first inequality again follows from the convexity of x^q. The last equality follows since $|H_i| = s_i^\gamma/(s'_\ell)^\gamma$. Thus, we have shown that the q^q-norm does not increase when jobs are moved from M' to M following the mapping $g'(\cdot)$. □

3 VECTOR SCHEDULING: MINIMIZING MAKESPAN

Using machine smoothing, we now give an $O\left(\frac{\log d}{\log \log d}\right)$-competitive algorithm for makespan minimization on (homogeneous) related machines (the first part of Theorem 1). By Lemma 4, we assume throughout that we have a smoothed instance, and only describe the algorithm to assign jobs to machine groups. Recall that each group is a set of identical machines, so we use the following theorem from [27] to assign jobs to individual machines in a group.

Theorem 6 ([27]). *For vector jobs scheduled on m identical machines, there is a deterministic algorithm that yields a schedule with makespan norm of*

$$O(\log d/\log d \log d) \cdot \left[\max_{k,j} p_j(k) + \max_k \sum_j p_j(k)/m \right].$$

This theorem implies that it is sufficient to find an assignment of jobs to machine groups that bounds the following function h against the optimal makespan norm opt:

$$h := \max_\ell \left(\max_{k, j \in J_\ell} \frac{p_j(k)}{s_\ell} + \max_k \frac{\sum_{j \in J_\ell} p_j(k)}{S(G_\ell)} \right),$$

where J_ℓ denotes the set of jobs assigned to group G_ℓ. Here, s_ℓ is the speed of any machine in G_ℓ, and $S(G_\ell)$ is the sum of speeds of all machines in G_ℓ.

Algorithm. Since all machines in a group have the same speed, we use s_ℓ to denote the speed of any machine in group G_ℓ. We assume that we know the value of the optimal makespan opt within a constant factor by using a standard guess-and-double technique. We say that a group G_ℓ is *permissible* for job j if $\max_k \frac{p_j(k)}{s_\ell} \leq$ opt, i.e., assigning the job alone does not violate the optimal bound. The algorithm assigns a job j to the group G_ℓ with the largest index ℓ (i.e., slowest speed) among all permissible groups for j.

Analysis. Let J_ℓ denote the jobs assigned to group G_ℓ. Since group G_ℓ is permissible for any job in J_ℓ, it follows that $\max_{k, j \in J_\ell} \frac{p_j(k)}{s_\ell} \leq$ opt. Furthermore, an optimal solution can schedule jobs in J_ℓ only on machines in $G_{\ell'}$ for $\ell' \leq \ell$, since G_ℓ is the slowest permissible group for jobs in J_ℓ. Thus, for any dimension k,

$$\sum_{j \in J_\ell} p_j(k) \leq \sum_{\ell'=0}^{\ell} S(G_{\ell'}) \cdot \text{opt}$$
$$= \left(S(G_\ell) + \sum_{\ell'=0}^{\ell-1} S(G_{\ell'}) \right) \cdot \text{opt} \leq 2 \cdot S(G_\ell) \cdot \text{opt}$$

where the last inequality uses **Property 2** of smoothed instances. Thus, $h \leq O(1) \cdot$ opt as desired.

4 SCALAR SCHEDULING: MINIMIZING q-NORMS

In this section, we describe our algorithm for optimizing arbitrary q-norms for the scalar scheduling problem. For convenience, we will work with the q^q-norm (i.e., the q-norm raised to the qth power) and give an $O(1)^q$-competitive algorithm, which is equivalent to an $O(1)$-competitive algorithm for the q-norm.

As in the previous section, we will assume throughout that we have a smoothed instance; by Lemma 5, this only loses a constant factor in the competitive ratio. Recall that the speed of a machine in group G_ℓ is denoted s_ℓ, and $p_{\ell j} := p_j/s_\ell$ denotes the processing time of job j on any machine in G_ℓ. Now, suppose a set of jobs J_ℓ is assigned to group G_ℓ. Then, two natural lower bounds on the q^q-norm of any assignment of these jobs to the individual machines in G_ℓ are given by: $|G_\ell| \cdot \left(\frac{\sum_{j \in J_\ell} p_{\ell j}}{|G_\ell|}\right)^q$, which is the q^q-norm of the optimal fractional assignment of these jobs, and $\sum_{j \in J_\ell} p_{\ell j}^q$, which is the q^q-norm of the optimal assignment if $|G_\ell| = \infty$. The next theorem, which follows from [9] and [24], states that the sum of these lower bounds can be attained in an online assignment on any set of identical machines. We provide a proof for sake of completeness.

Theorem 7 ([9], [24]). *For scalar jobs scheduled on m identical machines, there is a deterministic algorithm that yields a schedule with q^q-norm of* $O(1)^q \cdot \left[m \cdot \left(\frac{\sum_j p_j}{m}\right)^q + \sum_j p_j^q\right]$.

Proof. Our algorithm will be to simply schedule jobs greedily, i.e., for each job j that arrives, we schedule j on machine that currently has the smallest load. Let Λ be the load vector produced by the algorithm, and let Λ_i denote the load of ith machines. Let $\widehat{p_i}$ be the load of the last job that was assigned to machine i,and let Λ'_i be the load on machine without this last job (i.e., $\Lambda'_i = \Lambda_i - \widehat{p_i}$). Observe that

$$\|\Lambda\|_q^q = \sum_i \left(\Lambda'_i + \widehat{p_i}\right)^q \leq \sum_i \left(2\max(\Lambda'_i, \widehat{p_i})\right)^q$$
$$\leq 2^q \sum_i \left((\Lambda'_i)^q + \widehat{p_i}^q\right) \leq 2^q \left(m \cdot \left(\frac{\sum_j p_j}{m}\right)^q + \sum_j p_j^q\right),$$

as desired. The last inequality follows since the algorithm assigns greedily, and therefore $\Lambda'_i \leq \left(\sum_j p_j\right)/m$; we also clearly have that $\sum_i \widehat{p_i}^q \leq \sum_j p_j^q$. □

This theorem implies that it is sufficient for us to find an assignment of jobs to machine groups that bounds the following function h against the optimal q^q-norm:

$$h := \sum_\ell \left[|G_\ell| \cdot \left(\frac{\sum_{j \in J_\ell} p_{\ell j}}{|G_\ell|}\right)^q + \sum_{j \in J_\ell} p_{\ell j}^q\right].$$

Once this assignment is obtained, Theorem 7 is invoked separately on each machine group to obtain the final assignment of jobs to individual machines within each group.

The rest of this section describes the algorithm to assign jobs to machine groups in a smoothed instance. This algorithm has two parts: an online fractional algorithm that assigns jobs fractionally to machine groups, and an online rounding algorithm that converts this fractional assignment to an integral assignment.

4.1 Fractional Algorithm (A Continuous Greedy Algorithm)

We first define the fractional counterpart of the objective h defined above. Let $x_{\ell j}$ be the fraction of job j assigned to group G_ℓ in a fractional assignment. Then,

$$h(x) := \sum_\ell \left[|G_\ell| \cdot \left(\frac{\sum_{j \in J} p_{\ell j} \cdot x_{\ell j}}{|G_\ell|}\right)^q + \sum_{j \in J_\ell} p_{\ell j}^q \cdot x_{\ell j}\right].$$

For brevity of notation, we denote the *fractional load* of group G_ℓ by $\Lambda_\ell(x) := \sum_j \frac{1}{|G_\ell|} \cdot x_{\ell j} p_{\ell j}$ and separate the two terms in $h(x)$ as the *load-dependent* objective $f(x) := \sum_\ell |G_\ell| \cdot (\Lambda_\ell)^q$ and the *job-dependent* objective $g(x) := \sum_\ell \sum_j (p_{\ell j})^q \cdot x_{\ell j}$. We also call their sum $h(x)$ the *total* objective. The goal of the fractional algorithm is to obtain a fractional solution x that is $O(1)^q$-competitive for the total objective $h(x)$.

Algorithm. Our algorithm is inspired by a continuous greedy strategy on the objective $h(x)$. To define it precisely, we denote the two terms in the derivative $\frac{dh(x)}{dx_{\ell j}}$ by:

$$\alpha_{\ell j} := \frac{df(x)}{dx_{\ell j}} = |G_\ell| \cdot q \cdot (\Lambda_\ell)^{q-1} \cdot \frac{1}{|G_\ell|} \cdot p_{\ell j} = q \cdot (\Lambda_\ell)^{q-1} \cdot \frac{p_j}{s_\ell}$$
$$\beta_{\ell j} := \frac{dg(x)}{dx_{\ell j}} = (p_{\ell j})^q = \left(\frac{p_j}{s_\ell}\right)^q$$

The algorithm assigns an infinitesimal fraction of the current job j to the machine group G_ℓ that has the minimum value of $\eta_{\ell j} := \max(\alpha_{\ell j}, \beta_{\ell j})$. (Conceptually, this is similar to assigning to the machine that minimizes $\alpha_{\ell j} + \beta_{\ell j}$, which is the derivative of the objective $h(x)$, but using the max instead of the sum makes our analysis simpler.) In case of a tie between machine groups, the following rule is used:

- If there is a machine group G_ℓ with $\alpha_{\ell j} \leq \beta_{\ell j}$ among the tied groups, then this machine group is used for the assignment. Note that by **Property 3** of smoothed instances, no two machine groups can have an identical value of $\beta_{\ell j}$. It follows that there can only be at most one machine group among the ones tied on $\eta_{\ell j}$ that has $\eta_{\ell j} = \beta_{\ell j}$, i.e., $\alpha_{\ell j} \leq \beta_{\ell j}$. Hence, this rule is well-defined.
- If $\alpha_{\ell j} \geq \beta_{\ell j}$ for all tied machine groups, then we divide the infinitesimal job among the tied groups in proportion to $|G_\ell| \cdot s_\ell^\gamma$, where $\gamma = q/(q-1)$. These proportions are chosen to preserve the condition that the values of $\alpha_{\ell j}$ remain tied after the assignment (see Claim 8 below).

Analysis. Before giving any formal arguments, we first give some more intuition about the algorithm and a general overview of our analysis. Observe that if $\alpha_{\ell j} > \beta_{\ell j}$ for all machine groups G_ℓ throughout the algorithm, then the optimal strategy would be to keep all the $\alpha_{\ell j}$ identical, since otherwise, moving an infinitesimal fraction of j from a group with a higher value of $\alpha_{\ell j}$ to one with a lower value would improve the objective. This property is ensured by the second tie-breaking rule, which is stated in Claim 8 and can be verified by a simple calculation that we show later in the section.

Claim 8. *If job j is assigned in proportion to* $|G_\ell| \cdot s_\ell^\gamma$*, where* $\gamma = q/(q-1)$*, among machine groups* G_ℓ *with identical values of* $\alpha_{\ell j}$*, then the value of* $\alpha_{\ell j}$ *remains equal for these machine groups after the assignment.*

But, in general, it might so happen that $\beta_{\ell j} > \alpha_{\ell j}$ for a suffix of machine groups (recall that machine groups are ordered in decreasing speed, so the suffix represents slower machines). In this case, the algorithm might assign job j to faster groups even though this assignment makes the values of $\alpha_{\ell j}$ unequal for the different groups. But, in both these two types of assignments, the value of $\alpha_{\ell j}$ of a slower group never exceeds that of a faster group. Again we state this property in the next lemma and formally prove it in later in the section.

Lemma 9. *For any two groups* G_ℓ *and* $G_{\ell'}$ *with* $s_\ell > s_{\ell'}$*, and for any job j, it always holds that* $\alpha_{\ell j} \geq \alpha_{\ell' j}$.

This implies that for any two groups G_ℓ and $G_{\ell'}$ with $s_\ell > s_{\ell'}$ and any job j, α and β have exactly opposite relative orders throughout the algorithm: $\alpha_{\ell j} \geq \alpha_{\ell' j}$ (by Lemma 9) and $\beta_{\ell j} < \beta_{\ell' j}$ (by **Property 3** of smoothed instances). To get some more intuition about the algorithm, imagine extrapolating the discrete values of

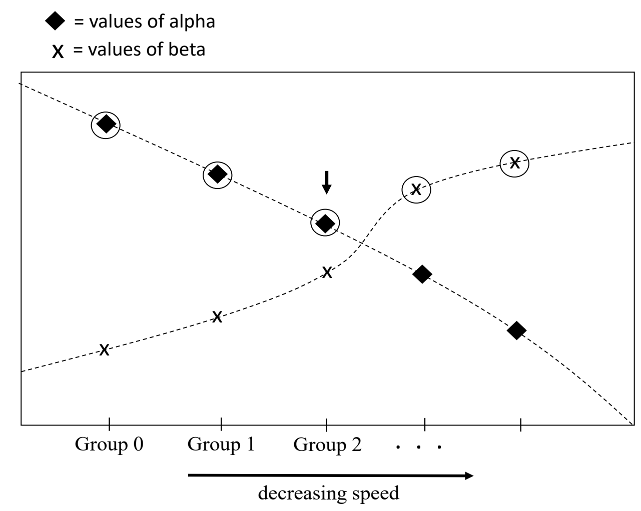

Figure 2: Monotonicity of $\alpha_{\ell j}$ and $\beta_{\ell j}$ values with machine speeds. The algorithm compares the circled values and chooses the machine group with the smallest of these values (in this case, it assigns to machine group 2). The dotted lines show an interpolation between the discrete speeds of machine groups that allows us to think of $\alpha_{\ell j}$ and $\beta_{\ell j}$ as continuous functions of machine speed. This interpolation is only for intuitive purpose, and is not formally used in the algorithm.

$\alpha_{\ell j}$ and $\beta_{\ell j}$ between the machine groups. Then, α is a monotone non-decreasing and β a monotone decreasing function of machine speed. The algorithm would ideally like to assign the fractional job to the point where these two curves cross, since it represents the minimum value of $\max(\alpha, \beta)$ across all speeds. But, since this crossing point may not represent an actual machine group, the algorithm compares the two machine groups that straddle this crossing point and assigns the fractional job to the group that minimizes $\max(\alpha, \beta)$ among these two machine groups. This is illustrated in Fig. 2.

Using the above interpretation that α and β are monotone functions of machine speed, we now sketch the rest of our analysis before giving it formally. We fix an optimal solution opt, and denote the fractional algorithm's solution by algo; let the corresponding fractional assignments be x_{opt} and x_{algo}. Now, for any assignment of a fractional job by algo, the same fractional job is assigned by opt to either the same machine group, or a slower one, or a faster one. It is easy to bound the objective due to assignments where algo and opt use the same machine group, so we only focus on the last two categories, which we call *red* and *blue* assignments respectively.

For red assignments, opt assigns to a slower machine group than algo; thus, its value of β is greater than the value of $\beta = \alpha$ for algo. Now, note that the job-dependent objective $g(x_{\text{opt}})$ is simply given by the sum over β of all assignments of opt, since β is independent of the current loads of machine groups. This allows us to bound the increase in the total objective $h(x_{\text{algo}})$ due to red assignments against the value of $g(x_{\text{opt}})$ (see Lemma 10).

Similarly, for blue assignments, opt assigns to a faster machine group than algo; thus, by Lemma 9, its value of α is greater than the value of $\alpha = \beta$ for algo. One would then hope to use a similar argument as above, that the value of $f(x_{\text{opt}})$ is the sum over α of all assignments of opt. Unfortunately, this is not true since the values of α are dependent on the current loads of algo, which may be different from the loads in opt. Instead, we need to use a more global argument for blue assignments. Note that if all assignments were blue, then using Claim 8, we can bound the value of $f(x_{\text{algo}})$ globally against the value of $f(x_{\text{opt}})$. There are two main challenges in generalizing this argument: first, it is possible that $\beta > \alpha$ for algo in a blue assignment, which makes bounding $f(x_{\text{algo}})$ insufficient, and second, red and blue assignments may be interleaved thereby invalidating the global argument that assumed all assignments to be blue. Overcoming these technical hurdles for blue assignments constitutes the bulk of our analysis (in particular see Lemma 11 for the final bound of the increase in $h(x_{\text{algo}})$ due to blue assignments).

This completes the overview of our analysis. To make the above arguments formal, we begin with giving proofs of Claim 8.

Proof of Claim 8. Recall that

$$\alpha_{\ell j} := q \cdot (\Lambda_\ell)^{q-1} \cdot \frac{p_j}{s_\ell} \tag{1}$$

Therefore its derivative with respect to an assignment $x_{\ell j}$ is:

$$\frac{d\alpha_{\ell j}}{dx_{\ell j}} = q(q-1) \cdot (\Lambda_\ell)^{q-2} \cdot \frac{p_j^2}{s_\ell^2} \cdot \frac{1}{|G_\ell|}.$$

Substituting for Λ_ℓ using (1) we have:

$$\frac{d\alpha_{\ell j}}{dx_{\ell j}} = q(q-1) \cdot \left(\frac{s_\ell \alpha_{\ell j}}{p_j \cdot q}\right)^{\frac{q-2}{q-1}} \cdot \frac{p_j^2}{s_\ell^2} \cdot \frac{1}{|G_\ell|} \tag{2}$$

To keep $\alpha_{\ell j}$ values equal while dividing $x_{\ell j}$ infinitesimally among the groups, we should assign mass inversely proportional to $\frac{d\alpha_{\ell j}}{dx_{\ell j}}$ to each group G_ℓ. However, since all G_ℓ already have equal $\alpha_{\ell j}$ upon the assignment, all terms in $\frac{d\alpha_{\ell j}}{dx_{\ell j}}$ except s_ℓ and $|G_\ell|$ are common across these groups. Thus, each group should receive mass in proportion to $s_\ell^{2-(q-2)/(q-1)}|G_\ell| = s_\ell^\gamma \cdot |G_\ell|$. □

Next, we give a formal proof of Lemma 9, which states that: *for any two groups G_ℓ and $G_{\ell'}$ with $s_\ell > s_{\ell'}$, and for any job j, it always holds that $\alpha_{\ell j} \geq \alpha_{\ell' j}$.*

Proof of Lemma 9. First, note that the lemma holds for all jobs if it does for any single job since $\alpha_{\ell j}/p_j$ is a function independent of job j. We now prove the lemma by showing that it inductively holds for the current job j at any time. For the sake of contradiction, suppose the property is violated for the first time by the current fractional assignment, which implies that $\alpha_{\ell j} = \alpha_{\ell' j}$ before the assignment and $\alpha_{\ell j} < \alpha_{\ell' j}$ after the assignment. Now, note that $\beta_{\ell' j} > \beta_{\ell j}$ by **Property 3** of smoothed instances. Therefore, the algorithm can make an assignment on $G_{\ell'}$ only if G_ℓ and $G_{\ell'}$ are tied with $\eta_{\ell j} = \alpha_{\ell j} = \alpha_{\ell' j} = \eta_{\ell' j}$. But in this case, by the second tie-breaking rule, the algorithm assigns job j to groups G_ℓ and $G_{\ell'}$ in proportion to $|G_\ell| \cdot s_\ell^\gamma$ and $|G_{\ell'}| \cdot s_{\ell'}^\gamma$, where $\gamma = q/(q-1)$. This assignment preserves $\alpha_{\ell j} = \alpha_{\ell' j}$ by Claim 8, which is a contradiction. □

Red and Blue Assignments. We now give the detailed analysis that bounds $h(x_{\text{algo}})$ against $h(x_{\text{opt}})$. We distinguish between two types of assignments, red and blue assignments that we precisely define below. (In the rest of the proof, we overload notation to denote a fractional job assigned in a single step of the fractional algorithm by j.) For a fractional job j, let opt(j) (resp., algo(j)) be the machine group on which it is assigned by opt (resp., algo). We call the assignment of a fractional job j a *red assignment* if opt assigns j on a slower machine group, i.e., if $s_{\text{opt}(j)} < s_{\text{algo}(j)}$; we call it a *blue assignment* if opt assigns j on a faster machine group, i.e., $s_{\text{opt}(j)} > s_{\text{algo}(j)}$. If opt($j$) = algo($j$) = G_ℓ, we call it a red assignment if $\beta_{\ell j} \geq \alpha_{\ell j}$ when the assignment was made; else, we call it a blue assignment.

We will analyze the total increase in the objective $h(x_{\text{algo}})$ caused by red and blue assignments separately. Note that there was a special case in the algorithm when machine groups were tied, where we assigned a fractional job to multiple machine groups. However, in this case, by **Property 2** of smoothed instances, at least half of the job is assigned to the slowest tied machine group. Since $\eta_{\ell j} = \alpha_{\ell j}$ for all tied groups in this case, the increase in $h(x)$ overall is at most a constant factor times the increase of $h(x)$ on the slowest machine group. Therefore, in this analysis, we will only consider the slowest machine group in this scenario. Thus, algo(j) is well-defined in all cases.

Analysis for Red Assignments. We first bound the contribution from red assignments.

Lemma 10. *The total increase in $h(x_{\text{algo}})$ due to red assignments of* algo *is at most twice the job-dependent objective $g(x_{\text{opt}})$ of* opt.

Proof. Consider a fractional job j that undergoes a red assignment in the algorithm. We have two cases. First, suppose $s_{\text{opt}(j)} < s_{\text{algo}(j)}$. Given that we only consider the assignment on the slowest group in case of a tie, we can conclude that:

$$\begin{aligned}\max(\alpha_{\text{opt}(j)j}, \beta_{\text{opt}(j)j}) = \eta_{\text{opt}(j)j} &> \eta_{\text{algo}(j)j}\\ &= \max(\alpha_{\text{algo}(j)j}, \beta_{\text{algo}(j)j}) \geq \alpha_{\text{algo}(j)j}\\ &\geq \alpha_{\text{opt}(j)j} \qquad \text{(by Lemma 9).}\end{aligned}$$

Therefore, $\beta_{\text{opt}(j)j} > \alpha_{\text{algo}(j)j}$. But, since $\beta_{\text{opt}(j)j} > \beta_{\text{algo}(j)j}$ by **Property 3** of smoothed instances, we have:

$$\alpha_{\text{algo}(j)j} + \beta_{\text{algo}(j)j} < 2\beta_{\text{opt}(j)j}.$$

Next, suppose opt(j) = algo(j). In this case,

$$\begin{aligned}\alpha_{\text{algo}(j)j} + \beta_{\text{algo}(j)j} &\leq 2\max(\alpha_{\text{algo}(j)j}, \beta_{\text{algo}(j)j})\\ &= 2\beta_{\text{algo}(j)j} = 2\beta_{\text{opt}(j)j},\end{aligned}$$

where the second to last equality follows from the definition of red assignments. To complete the proof of the lemma, we note that the sum of $\beta_{\text{opt}(j)j}$ across all fractional jobs is precisely equal to $g(x_{\text{opt}})$. □

Analysis for Blue Assignments. We are left to bound the total increase in $h(x_{\text{algo}})$ due to blue assignments. For blue assignments, opt assigns the fractional jobs to faster machine groups than algo. Our goal is to show the following lemma.

Lemma 11. *The total increase in objective due to blue assignments in* algo *is at most $O(1)^q$ times the load-dependent objective of* opt.

Using an argument similar to above lemma, we can bound the increase in $h(x_{\text{algo}})$ due to blue assignments against the value of $\alpha_{\text{opt}(j)j}$. But, unlike the fact that the sum of $\beta_{\text{opt}(j)j}$ across all jobs gives the value of $g(x_{\text{opt}})$, adding the values of $\alpha_{\text{opt}(j)j}$ across all jobs does not yield the value of $f(x_{\text{opt}})$. This is because $\alpha_{\text{opt}(j)j}$ are defined based on algo's current loads, which may be different from opt's loads, whereas $\beta_{\text{opt}(j)j}$ is load-independent.

To understand the intuition behind our analysis of blue assignments, let us imagine an idealized scenario where $\alpha_{\ell j} > \beta_{\ell j}$ throughout the algorithm. In this case, by the second tie-breaking rule, algo maintains equal values of $\alpha_{\ell j}$ across all machine groups ℓ for all jobs j. Note that this is an optimal fractional assignment for the load-dependent objective $f(x)$; therefore, $f(x_{\text{algo}}) \leq f(x_{\text{opt}})$. The same argument works even if $\alpha_{\ell j}$ is not equal for all groups, provided all jobs are blue, by replacing uniformity of $\alpha_{\ell j}$ by the monotonicity property from Lemma 9. However, there are two main difficulties with generalizing this argument further:

1. First, for a blue assignment of job j to machine group algo(j), it may be the case that $\beta_{\text{algo}(j)j} > \alpha_{\text{algo}(j)j}$. In this case, we need to bound $\beta_{\text{algo}(j)j}$ and not $\alpha_{\text{algo}(j)j}$ in order to bound the increase in the total objective $\alpha_{\text{algo}(j)j} + \beta_{\text{algo}(j)j}$.
2. Second, we need to account for the fact that not all assignments are blue, and a general instance can interleave red and blue assignments.

To address the first issue, we specifically consider the blue assignments with $\beta_{\text{algo}(j)j} > \alpha_{\text{algo}(j)j}$; let us call them *special* assignments. For all such special assignments, we modify algo to algo′ by additionally assigning the fractional job to the machine group (denoted algo$(j)^+$) that is immediately faster than algo(j). The idea behind this additional dummy assignment is that $\alpha_{\text{algo}(j)^+j} \geq \eta_{\text{algo}(j)j}$ irrespective of whether $\eta_{\text{algo}(j)j} = \beta_{\text{algo}(j)j}$ or $\eta_{\text{algo}(j)j} = \alpha_{\text{algo}(j)j}$. Therefore, for the special assignments, we can bound the increase in the total objective of algo by $\alpha_{\text{algo}(j)^+j}$ due to the dummy assignments that we added. Correspondingly, we modify opt to opt′ by adding a second copy of each such fractional job to opt(j). Note that for special blue assignments, we have the strict inequality $s_{\text{opt}(j)} > s_{\text{algo}(j)}$; else, we would call it a red assignment. Hence, these additional dummy assignments are also blue assignments. We will establish that these modifications do not significantly change the objectives of the two solutions algo and opt (Lemmas 12 and 13).

To handle the second issue, we modify opt′ further to opt″ by adding the load due to red assignments in algo to each machine group in opt. This allows us to view the red assignments as blue assignments for the purposes of this analysis, since opt″ now has a copy of every red job on the same machine group as algo. Again, we establish that this transformation does not significantly change the objective of opt′ (Lemma 14).

Once we have modified algo to algo′ and opt to opt″ respectively, we are able to show a strong property of the load profiles of algo′ and opt″, namely that for *any* prefix of machine groups starting with the fastest group, the total load of opt″ is at least as much as that of algo′. Informally, this means that opt″ is even more biased than algo′ in terms of assigning jobs to faster machines. But, note that Lemma 9 asserts that algo is already biased toward faster machines than the optimal allocation for the load-dependent objective $f(x)$, and it turns out, so is algo′. Combining these facts, we bound the increase in the total objective of algo′ against the load-dependent objective of opt″ (Lemma 16).

We now give a formal proof for the increase in the total objective of algo due to blue assignments, based on the outline presented above. Recall that there are three parts to this analysis: (a) modifying algo to algo′ and opt to opt′ respectively do not significantly change their objectives, (b) modifying opt′ to opt″ does not significantly change its objective, and (c) the increase in the total objective of algo′ due to blue assignments can be bounded against the load-dependent objective of opt″.

First, we show that in handling the first issue, modifying algo to algo′ and opt to opt′ respectively do not significantly change the objectives of the respective solutions. The first lemma is immediate.

Lemma 12. *The load-dependent objective $f(x_{\text{opt}'})$ in* opt′ *is at most 2^q times the corresponding objective $f(x_{\text{opt}})$ in* opt.

Lemma 13. *The total objective $h(x_{\text{algo}})$ due to blue assignments in* algo *is at most twice the load-dependent objective $f(x_{\text{algo}'})$ due to blue assignments in* algo′ .

Proof. We consider two cases. First, suppose $\alpha_{\text{algo}(j)j} \geq \beta_{\text{algo}(j)j}$. This is not a special blue assignment. In this case,

$$\alpha_{\text{algo}(j)j} + \beta_{\text{algo}(j)j} \leq 2\alpha_{\text{algo}(j)j}.$$

Since algo′ has at least as much load on every machine group as algo, it follows that the total increase of objective in algo due to assignments in this case is at most twice the load-dependent objective of algo′.

Next, suppose $\alpha_{\text{algo}(j)j} < \beta_{\text{algo}(j)j}$ in a blue assignment. This is a special blue assignment, and we have $s_{\text{opt}(j)} > s_{\text{algo}(j)}$, as noted earlier. In this case, $\beta_{\text{algo}(j)^+j} < \beta_{\text{algo}(j)j}$, but $\eta_{\text{algo}(j)^+j} \geq \eta_{\text{algo}(j)j}$. Therefore, $\alpha_{\text{algo}(j)^+j} \geq \beta_{\text{algo}(j)j}$ and $\alpha_{\text{algo}(j)^+j} \geq \alpha_{\text{algo}(j)j}$. Therefore, we have

$$\alpha_{\text{algo}(j)j} + \beta_{\text{algo}(j)j} \leq 2\alpha_{\text{algo}(j)^+j}.$$

But, for every special assignment to machine group algo(j) in algo, there is a corresponding assignment to group algo$(j)^+$ in algo′. Therefore, the total increase of objective in algo due to special assignments is at most twice the load-dependent objective of algo′. □

Now, we show that in handling the second issue, modifying opt′ to opt″ does not significantly change the objective.

Lemma 14. *The load-dependent objective $f(x_{\text{opt}''})$ in* opt″ *is at most 2^q times the load-dependent objective $f(x_{\text{opt}'})$ in* opt′ *plus 2^{q+1} times the job-dependent objective $g(x_{\text{opt}})$ in* opt.

Proof. We classify machine groups into two groups. The first type of group is one where the load in opt′ is at least its load from red assignments in algo. The load in opt″ for such groups is at most twice the load in opt′. Therefore for these machine groups, the load-dependent objective in opt″ is at most 2^q times load-dependent objective in opt′.

The second type of machine group is one where the red load in algo is more than the load in opt′. The load in opt″ for such machine groups is at most twice the red load in algo. Therefore by Lemma 10, the load-dependent objective in opt″ is at most $2 \cdot 2^q$ times the job-dependent objective $g(x_{\text{opt}})$ in opt. □

We will now be able to apply our high level approach and show that the load-dependent objective of algo′ is bounded by that of opt″. We first show the following lemma on load profiles, which formalizes our earlier intuition.

Lemma 15. *Consider two load profiles ψ and ξ over the machine groups with the following properties:*

1. *(First condition) For any prefix $\mathcal{G}$ of machine groups in decreasing order of speeds, the total job volumes satisfy: $\sum_{G_\ell \in \mathcal{G}} \psi_\ell \cdot |G_\ell| \cdot s_\ell \geq \sum_{G_\ell \in \mathcal{G}} \xi_\ell \cdot |G_\ell| \cdot s_\ell$.*
2. *(Second condition) There exists a $\mu \leq 1$ such that for any two machine groups G_ℓ and $G_{\ell'}$, we have:*

$$\frac{\xi_\ell^{q-1}}{s_\ell} \cdot \frac{1}{|G_\ell|} \geq \mu \cdot \frac{\xi_{\ell'}^{q-1}}{s_{\ell'}} \cdot \frac{1}{|G_{\ell'}|}.$$

Then, the load-dependent objective of load profile ψ is at least $\mu^{\frac{q}{q-1}}$ times the load-dependent objective of load profile ξ.

Proof. First, we transform the load profile ξ to χ so as to change the value of μ to 1 in the second condition. For any group G_ℓ, we set χ_ℓ so that it satisfies

$$\frac{\chi_\ell^{q-1}}{s_\ell} \cdot \frac{1}{|G_\ell|} = \min_{G_{\ell'}: s_{\ell'} \geq s_\ell} \frac{\xi_{\ell'}^{q-1}}{s_{\ell'}} \cdot \frac{1}{|G_{\ell'}|}.$$

Since $\chi_\ell \leq \xi_\ell$ for any machine group G_ℓ, the first condition holds for ψ and χ as well. Furthermore, by definition of χ, it satisfies the second condition with $\mu = 1$. Finally, note that by the second condition on ξ,

$$\frac{\chi_\ell^{q-1}}{s_\ell} \cdot \frac{1}{|G_\ell|} = \min_{G_{\ell'}: s_{\ell'} \geq s_\ell} \frac{\xi_{\ell'}^{q-1}}{s_{\ell'}} \cdot \frac{1}{|G_{\ell'}|} \geq \mu \cdot \frac{\xi_\ell^{q-1}}{s_\ell} \cdot \frac{1}{|G_\ell|}. \quad (3)$$

Now, we use an exchange argument to transform ψ without increasing its load-dependent objective until for every machine group G_ℓ, we have $\psi_\ell \geq \chi_\ell$. In each step of the exchange, we identify the slowest machine group G_ℓ where $\psi_\ell < \chi_\ell$. By the first condition, there must be a machine group $G_{\ell'}$ with $s_{\ell'} > s_\ell$ such that $\psi_{\ell'} > \chi_{\ell'}$ and for every prefix $\mathcal{G}$ of machine groups in decreasing order of speeds containing $G_{\ell'}$ but not containing G_ℓ, the following strict inequality holds:

$$\sum_{G_\ell \in \mathcal{G}} \psi_\ell \cdot |G_\ell| \cdot s_\ell > \sum_{G_\ell \in \mathcal{G}} \chi_\ell \cdot |G_\ell| \cdot s_\ell. \quad (4)$$

Furthermore, using the second condition (with now $\mu = 1$), we have that

$$\frac{\psi_{\ell'}^{q-1}}{s_{\ell'}} \cdot \frac{1}{|G_{\ell'}|} > \frac{\chi_{\ell'}^{q-1}}{s_{\ell'}} \cdot \frac{1}{|G_{\ell'}|} \geq \frac{\chi_\ell^{q-1}}{s_\ell} \cdot \frac{1}{|G_\ell|} > \frac{\psi_\ell^{q-1}}{s_\ell} \cdot \frac{1}{|G_\ell|}. \quad (5)$$

Now, we move an infinitesimal job volume from group $G_{\ell'}$ to group G_ℓ in ψ. Inequality (5) implies that the load-dependent objective of ψ decreases due to this move. Furthermore, both conditions of the lemma continue to remain valid by Eqns. (4) and (5). Such moves are repeatedly performed to obtain a load profile ψ' with at most the load-dependent objective of ψ, but additionally satisfying $\psi'_\ell \geq \chi_\ell$ for all machine groups G_ℓ.

At this point, the lemma holds for the transformed load profile χ with $\mu = 1$. To translate this back to the original load profile ξ, note that Eqn. (3) implies that $\chi_\ell \geq \mu^{1/(q-1)} \cdot \xi_\ell$ for every machine group G_ℓ. □

We now apply Lemma 15 to algo′ and opt′′ to get our desired bound.

Lemma 16. *The load-dependent objective of* algo′ *is at most* 2^q *times the load-dependent objective of* opt′′.

Proof. In Lemma 15, we set ψ to the load profile of opt′′ and ξ to the load profile of algo′.

The first condition of Lemma 15 follows from the following observations: (a) for blue assignments in algo, $s_{\text{opt}(j)} \geq s_{\text{algo}(j)}$; (b) for red assignments in algo, the same fractional job j is assigned to algo(j) in transforming opt′ to opt′′; (c) finally, for special assignments added in transforming algo to algo′, we have $s_{\text{opt}(j)} > s_{\text{algo}(j)}$, i.e., $s_{\text{opt}(j)} \geq s_{\text{algo}(j)^+}$.

We now check the second condition of Lemma 15. From Lemma 9, the condition holds with $\mu = 1$ for algo. In algo′, the load Λ_{ℓ^+} on a machine group ℓ increases by the total load due to special assignments on machine group G_ℓ, i.e., by at most $\Lambda_\ell \cdot \frac{s_\ell}{s_{\ell^+}} \leq \Lambda_\ell$. But, by Lemma 9, $\Lambda_\ell \leq \Lambda_{\ell^+}$. Therefore, the load on machine group G_{ℓ^+} increases by at most a factor of 2. It follows that the second condition of Lemma 15 holds with $\mu = 1/2^{q-1}$.

Now, the lemma follows by applying Lemma 15. □

Combining Lemmas 12, 13, 14, and 16, we obtain the desired bound for blue assignments, i.e., prove Lemma 11.

Lemmas 11 and 10 imply that the algorithm is $O(1)^q$-competitive on objective $h(x)$, as desired.

4.2 Rounding Algorithm

In this section, we give an online rounding procedure that converts a fractional assignment to an integral assignment for machine groups with a loss of $O(1)^q$ in the total objective $h(x)$.

Algorithm. Recall that $x_{\ell j}$ denotes the fraction of job j assigned to machine group G_ℓ, where the machine groups are denoted $G_0, G_1, \ldots$ from the fastest to the slowest group. Let $m(j)$ denote the *median* of job j, which we define to be the machine group that satisfies $\sum_{\ell \leq m(j)} x_{\ell j} \geq 1/2$ and $\sum_{\ell \geq m(j)} x_{\ell j} \geq 1/2$. In our integral assignment, we assign job j to group $G_{m(j)}$.

Analysis. Let x' be the solution produced by the rounding algorithm. We will show the following lemma.

Lemma 17. *After rounding x to x', we have $h(x') \leq O(1)^q \cdot h(x)$.*

We consider the load-dependent and job-dependent objectives $f(x')$ and $g(x')$ separately.

Lemma 18. $f(x') \leq 4^q \cdot f(x)$.

Proof. We perform two transformations on assignment x such that $f(x)$ can only increase by a factor of at most 4^q, and then compare the resulting assignment with x'.

Our first transformation converts x to an assignment y, as follows. For each job j, we set $y_{\ell j} := 2x_{\ell j}$ for $\ell \leq m(j) - 1$. We then allocate the remainder of the job to group $G_{m(j)}$, i.e., $y_{m(j)j} := 1 - \sum_{\ell \leq m(j)-1} y_{\ell j}$. For the rest of the groups $\ell \geq m(j) + 1$, we set $y_{\ell j} := 0$. Observe that based on the definition of $m(j)$, this is always possible. It is straightforward to see that $f(y) \leq 2^q f(x)$ since for any given group, we are at most doubling its load.

Then, we construct an optimal prefix-constrained schedule, y' w.r.t. f. We say that a schedule is prefix-constrained if each job j's assignment is restricted to machine groups $G_0, G_1, \ldots G_{j(m)}$. Note that y is prefix-constrained. Therefore, $f(y') \leq f(y)$. We construct y' as follows. We consider machines groups in increasing order of their speeds. For a group $G_{\ell'}$ in consideration, we consider each job j with $m(j) = \ell'$. We continuously assign job j to the machine group ℓ with the minimum $\alpha_\ell := q(\Lambda_\ell)^{q-1}/s_\ell$ (or equivalently to $\alpha_{\ell,j}$). Note that α_ℓ has no dependency on p_j unlike $\alpha_{\ell,j}$. Since we only consider the load-dependent objective, it is an easy exercise to show that this continuous process yields an optimal assignment w.r.t. f. Formally, one can show that the monotonicity of α values, i.e., $\alpha_0 \geq \alpha_1 \geq \ldots$, holds throughout. The optimality follows from this monotonicity property and the fact that two machine groups continue to have the same α_ℓ values once they do.

We observe that at least half of each job j is assigned to $G_{m(j)}$ in y'. Due to the monotonicity property, j is assigned to machine group $m(j)$ least when $\alpha_0 = \alpha_1 = \alpha_2 = \ldots = \alpha_{m(j)}$ before j starts getting assigned. In this case, job j is assigned to machine group G_ℓ in proportion to $S(\ell)$, which implies $G_{m(j)}$ gets at least half of the job j since $S(G_{m(j)}) \geq S(G_0) + S(G_1) + S(G_2) + \ldots + S(G_{j(m)-1})$ (**Property 2**).

Finally, we obtain the final assignment x' from y' by letting each job j to be fully assigned to $G_{m(j)}$. This increases each machine group's load by a factor of at most 2, therefore we have $f(x') \leq 2^q \cdot f(y')$. □

Lemma 19. $g(x') \leq 2 \cdot g(x)$.

Proof. Consider a transformation of x to y where for each job j, we set $y_{\ell j} := 2x_{\ell j}$ for $\ell \geq m(j)+1$. We then allocate the remainder of the job to group $G_{m(j)}$, i.e., $y_{m(j)j} := 1 - \sum_{\ell \geq m(j)+1} y_{\ell j}$. For the rest of the groups $\ell \leq m(j)-1$, we set $y_{\ell j} := 0$. Since $\sum_{\ell \geq m(j)} x_{\ell j} \geq 1/2$, for each job j and machine group G_ℓ, we have $y_{\ell j} \leq 2x_{\ell j}$. Thus, $g(y) \leq 2 \cdot g(x)$. On the other hand, since every job j is assigned to $G_{m(j)}$ in x' and to machine groups that are no faster than $G_{m(j)}$ in y, it follows that $g(x') \leq g(y)$. □

Combining Lemmas 18 and 19, we obtain Lemma 17.

5 VECTOR SCHEDULING: MINIMIZING q-NORMS

As in the previous section on scalar scheduling, we will assume throughout that we have a smoothed instance; by Lemma 5, this only loses a constant factor in the competitive ratio. We also assume that $q \geq c \log d$ for a large enough constant c, since otherwise, we can use a scheduling algorithm for unrelated machines in [27] to find an $O(\log d + q) = O(\log d)$-competitive assignment. As for the scalar case, we derive two natural lower bounds on the q^q-norm of any assignment of these jobs to the individual machines in G_ℓ. The first bound is $\max_k |G_\ell| \cdot \left(\frac{\sum_{j \in J_\ell} p_{\ell j}(k)}{|G_\ell|}\right)^q$, which is the q^q-norm of the optimal fractional assignment of these jobs, and the second bound is $\max_k \sum_{j \in J_\ell} \left(p_{\ell j}(k)\right)^q$, which is the q^q-norm of the optimal assignment if $|G_\ell| = \infty$. Then,

$$h := \sum_k \left[\sum_\ell |G_\ell| \cdot \left(\frac{\sum_{j \in J_\ell} p_{\ell j}(k)}{|G_\ell|} \right)^q + \sum_\ell \sum_{j \in J_\ell} \left(p_{\ell j}(k) \right)^q \right]$$

lower bounds the optimal value of the q^q-norm up to a factor of $d = O(1)^q$ (recall than $q = \Omega(\log d)$) since we replaced the max over dimensions k with a sum over dimensions k.

The next theorem, which follows from previous work [27], asserts that this lower bound can be achieved on a set of identical machines up to a factor of $O\left(\frac{\log d}{\log \log d}\right)^{q-1}$.

Theorem 20 ([27]). *For vector jobs scheduled on m identical machines, there is a deterministic algorithm that yields a schedule such that the maximum q^q-norm across dimensions is at most $O\left(\frac{\log d}{\log \log d}\right)^{q-1}$ ·* $\sum_k \left[m \cdot \left(\frac{\sum_j p_j(k)}{m} \right)^q + \sum_j \left(p_j(k) \right)^q \right]$.

This theorem implies that it is sufficient to find an assignment of jobs to machine groups that bounds function h against the optimal q^q-norm. Once this assignment is obtained, Theorem 20 is invoked separately on each machine group to obtain the final assignment of jobs to individual machines within each group.

The rest of this section describes the algorithm to assign jobs to machine groups in a smoothed instance.

5.1 Algorithm Definition

Our algorithm assigns job in two phases. In Phase 1, we define a single scalar load derived from $\max_k p_j(k)$ and assign it using the scalar algorithm for q norms given in Section 4. This produces an assignment that we call the *scalar solution*. Using the scalar solution, we then determine a set of *candidate groups* $\mathcal{G}_j$ to which job j can be assigned in the second phase; we call such assignments *restricted assignments*. In Phase 2, we obtain a competitive restricted assignment, which is the assignment the algorithm outputs. We now describe these phases in more detail.

Phase 1: Scalar Solution. Let $p_{j,\max} := \max_k p_j(k)$. We set the scalar size of job j to be $p_{j,\max}$ and use the algorithm for scalar jobs from Section 4.

Let $G_{m(j)}$ be the group where job j is assigned in the scalar solution (we use the notation $m(j)$ since recall that in our scalar rounding algorithm, we assign a job to the median group of job j from the fractional assignment). Then, the set of candidate groups for job j, denoted $\mathcal{G}_j$, is a collection of $O(\log d)$ consecutive groups containing $G_{m(j)}$ given by:

$$\mathcal{G}_j := \{G_{\max\{0, m(j)-4\log d\}}, \ldots, G_{\min\{M, m(j)+4\log d\}}\}.$$

(Here, the total set of groups is $G_0, G_1, \ldots, G_M$.)

Phase 2: Restricted Assignment. We schedule jobs with the same candidate groups separately. Namely, fix a set of candidate groups $\mathcal{G}$ and let $\text{opt}_{\mathcal{G}}$ be the optimal (w.r.t. h) schedule for *only* the jobs with candidate groups $\mathcal{G}$ *on the groups in* $\mathcal{G}$. Our goal will be to find a solution that satisfies the following set of constraints:

$$\max_k \max_{G_\ell \in \mathcal{G}} \sum_{j \in J_\ell} \frac{p_{\ell j}(k)}{|G_\ell|} \leq \text{opt}_{\mathcal{G}}^{1/q} \quad \text{and} \qquad (6)$$
$$\max_{G_\ell \in \mathcal{G}} \sum_k \sum_{j \in J_\ell} (p_{\ell j}(k))^q \leq \text{opt}_{\mathcal{G}}.$$

Note that $\text{opt}_{\mathcal{G}}$ satisfies these conditions. We assume that $\text{opt}_{\mathcal{G}}$ is known to the algorithm, which is w.l.o.g. by a standard guess-and-double technique.

We interpret this online problem as makespan minimization for unrelated machines, i.e., we think of each group G_ℓ as a meta-machine and of each job j as having a load of $\frac{p_{\ell j}(k)}{|G_\ell|}$ on a meta-machine G_ℓ in dimension k. We also create a special dimension 0 to encode the second set of constraints: job j has load $\sum_k (p_{\ell j}(k))^q$ on meta-machine G_ℓ on dimension 0. Then, the problem is reduced to finding an assignment where the makespan on dimension 0 is upper bounded by $\text{opt}_{\mathcal{G}}$, and the makespan on other dimensions from 1 to d is upper bounded by $\text{opt}_{\mathcal{G}}^{1/q}$. In [27], this problem was studied under the name of any norm minimization for unrelated machines (VSANY-U). Using the algorithm in [27], one can find a solution simultaneously minimizing the $\log(|\mathcal{G}|)$-norm on each dimension with the target values $\text{opt}_{\mathcal{G}}^{1/q}$ on dimensions $1, 2, 3, \ldots, d$, and $\text{opt}_{\mathcal{G}}$ on dimension 0, which is equivalent to the makespan optimization problem that we have defined, up to a constant factor.

5.2 Analysis

At a high level, our analysis comprises of two steps. First, we show in the following lemma that by restricting our assignment in Phase 2, we do not lose more than a constant factor with respect to an unrestricted assignment.

Lemma 21. *The optimal fractional restricted assignment is at most $O(1)^q$ times the optimal unrestricted assignment with respect to objective h.*

Next, we show that for the restricted assignment problem, our algorithm has a competitive ratio of $O(\log^2 d)^q$.

Lemma 22. *For the restricted assignment problem defined in Phase 2, our algorithm has a competitive ratio of $O(\log^2 d)^q$.*

Combining the above two lemmas gives us the overall competitive ratio of $O(\log^2 d)^q$ for the algorithm on objective h. We prove these two lemmas in the following sections.

5.2.1 *Proof of Lemma 21.*

In this section we prove Lemma 21. Our high-level approach will be to construct a fractional assignment, denoted frac_r, and show that this fractional assignment has objective at most $O(1)^q$ times that of the optimal assignment opt with respect to h. Additionally, frac_r will be constructed in a way such that we can obtain an integral assignment of similar objective by using the rounding algorithm in Section 4. Our analysis will imply that using this rounding only increases the objective of the solution by at most $O(1)^q$ factor, thus proving the lemma. We note that to simplify our notation, we may let opt and frac_r also denote their objective values, depending on context.

First, we show that we can make a certain assumption on the sizes of jobs across all dimensions without changing the objective of instance by too much. Recall that $p_{j,\max} := \max_k p_j(k)$. We first observe that we can assume wlog that each job j has size at least $\frac{p_{j,\max}}{d^2}$ on all dimensions.

Lemma 23. *If we increase each job j's size so that j has size $\max\{p_j(k), \frac{p_{j,\max}}{d^2}\}$ on each dimension k, then objective h increases by a factor of at most 2^q.*

Proof. Let $x_{\ell j}$ be the fraction of job j assigned to group ℓ. (So note that this lemma also holds for fractional assignments). Also, recall our definitions of load- and job-dependent objectives from Section 4. We first consider the job-dependent objective. Before the change, j's contribution to the objective for group G_ℓ is $\sum_k (p_{\ell j}(k))^q x_{\ell j}$. After the change, this quantity increases by at most

$$\sum_k (p_{j,\max}/d^2)^q x_{\ell j} \leq \max_k (p_{j,\max})^q x_{\ell j} \leq \sum_k (p_{\ell j}(k))^q x_{\ell j}.$$

Thus, the job-dependent objective at most doubles.

Next, we consider the load-dependent objective. Fix a group G_ℓ. Recall that $\Lambda_\ell(k) := \sum_j (p_{\ell j}(k) x_{\ell j})$. Thus, group G_ℓ's contribution to the load-dependent objective is $|G_\ell|^{1-q} \sum_k (\Lambda_\ell(k))^q$. We will show that $\sum_k (\Lambda_\ell(k))^q$ increases by a factor of at most 2^q after the job size change. Specifically, after the job size change, each $\Lambda_\ell(k)$ value increases by at most $\sum_{k' \neq k} \Lambda_\ell(k')/d^2 \leq \sum_{k'} \Lambda_\ell(k')/d^2$. Thus, after the change, $\sum_k (\Lambda_\ell(k))^q$ is upper bounded by

$$\sum_k \left(\Lambda_\ell(k) + \sum_{k'} \frac{\Lambda_\ell(k')}{d^2}\right)^q \leq \sum_k 2^{q-1} \left((\Lambda_\ell(k))^q + \left(\sum_{k'} \frac{\Lambda_\ell(k')}{d^2}\right)^q\right)$$
$$\leq 2^{q-1} \sum_k (\Lambda_\ell(k))^q + \frac{d \cdot 2^{q-1}}{d^{2q}} \sum_{k'} (\Lambda_\ell(k'))^q \leq 2^q \sum_k (\Lambda_\ell(k))^q.$$

Thus, we have shown that each of the job-dependent and load-dependent objectives increases by a factor of at most 2^q, proving the lemma. □

By Lemma 23, we can assume wlog that jobs have these new sizes after the change. We now make additional transformations to the instance. One transformation is applied to the instance that the optimal solution processes, while the other is applied to the instance that $\mathrm{frac_r}$ is defined on.

Specifically, let opt denote a fixed optimal assignment with respect to h. We say that an input is *uniform* if every job has equal size in each dimension. We will consider two uniform inputs derived from the original input. Let $J_{\max}$ denote the set of jobs where each job j's size vector is replaced with $p_{j,\max} \cdot \langle 1, 1, \ldots, 1\rangle$. Similarly, let $J_{\min}$ denote the set of jobs where each job j's size vector is replaced with $\frac{p_{j,\max}}{d^2} \cdot \langle 1, 1, \ldots, 1\rangle$. Note that $J_{\max}$ is as hard as the original input (i.e., the objective of any solution to $J_{\max}$ upper bounds the corresponding solution on the original instance). Similarly, $J_{\min}$ is as easy as the original input. Since our goal is to upper bound $\mathrm{frac_r}$ by opt, we assume that $\mathrm{frac_r}$ has to process $J_{\max}$ while opt processes $J_{\min}$. Since all jobs have uniform sizes, all dimensions have an equal contribution to the objective. Therefore, for the rest of the analysis we can treat each solution as a scalar solution with these corresponding job sizes. Also for simplicity, we denote $p_{j,\max}$ as p_j for the rest of the section.

We are now ready to construct the solution $\mathrm{frac_r}$. We categorize jobs into three groups based on how they are assigned in opt. We say that a job j is red (blue, resp.) if it is assigned in opt to a group slower (faster, resp.) than the candidate groups in $\mathcal{G}_j$; otherwise, the job is grey. For each grey job j, we simply assign j in $\mathrm{frac_r}$ to the group its assigned to in opt. To assign red and blue jobs, we use the fractional assignment that was used to produce the assignment from Phase 1 of the algorithm (the fractional assignment obtained before applying the rounding algorithm; call this solution $\mathrm{algo_{frac}}$). In particular, for red jobs j, we assign j identically to $\mathrm{algo_{frac}}$ for $G_{m(j)}$ and all faster groups. Similarly, blue jobs j are assigned identically to $\mathrm{algo_{frac}}$ for $G_{m(j)}$ and slower machines. We note that the blue and red jobs may not be completely assigned (i.e., we may only fractionally assign half of each of these jobs), and so in some sense $\mathrm{frac_r}$ is not a complete/valid assignment. However, since on doubling the fractional assignment, all jobs will increase objective h by at most 2^q, it is easy to convert $\mathrm{frac_r}$ into a valid assignment with a comparable objective.

We decompose the objective to analyze the contribution of jobs of each type separately. Let BLUE, RED, GREY denote set of blue, red, and grey jobs, respectively. Also let $\mathrm{frac_r}^{\mathrm{BLUE}}$, $\mathrm{frac_r}^{\mathrm{RED}}$, and $\mathrm{frac_r}^{\mathrm{GREY}}$ denote the objective h when only considering the assignments of blue, red, and grey jobs respectively in $\mathrm{frac_r}$. Given these definitions, it is straightforward to show the following decomposition.

Lemma 24. $\mathrm{frac_r} \leq 3^q(\mathrm{frac_r}^{BLUE} + \mathrm{frac_r}^{RED} + \mathrm{opt})$.

Therefore, for the rest of the analysis we are left with bounding $\mathrm{frac_r}^{\mathrm{BLUE}}$ and $\mathrm{frac_r}^{\mathrm{RED}}$. To simplify our presentation, we assume that each red and blue job is broken up into infinitesimal sized jobs; namely, we will assume that a red/blue infinitesimal job j is fully assigned when $\sum_\ell x_{\ell j} = \delta$ for an infinitesimally small value $\delta > 0$. This modification can be done by replacing each job j by a set of jobs $j_1, j_2, \ldots, j_{1/\delta}$ with job size $\delta \cdot p_j$ such that $\sum_\ell x_{\ell j_r} = \delta$ for each newly created job j_r. Note that this alternate view is valid since red and blue jobs are assigned fractionally in $\mathrm{frac_r}$.

Observe that since red and blue jobs are assigned based on $\mathrm{algo_{frac}}$, an infinitesimal job j can be assigned to multiple groups. Let $G_{e(j)}$ be the slowest group to which j is assigned in $\mathrm{frac_r}$. Recall that in Section 4 we observed that at least half of any infinitesimal job j is assigned to group $G_{e(j)}$. Therefore, we can assume that each infinitesimal job j is only assigned to $G_{e(j)}$, since this modification will only increase objective h by at most a factor of at most 2^q.

Let J_e denote the set of infinitesimal jobs j with $e(j) = e$. Note that $\{J_e\}_e$ is a partition of jobs. We are now ready to bound $\mathrm{frac_r}^{\mathrm{RED}}$ and $\mathrm{frac_r}^{\mathrm{BLUE}}$.

Lemma 25. $\mathrm{frac_r}^{RED} \leq \mathrm{opt}$.

Proof. Fix a group G_e. Consider any infinitesimal job $j \in J_e \cap$ RED. The job was assigned to G_e, but not to any slower groups since $\alpha_{ej} < \beta_{e+1j}$. This implies that the increase of the load-dependent objective due to j's assignment is smaller than the analogous increase of the job-dependent objective. Thus, it suffices to show that $\mathrm{frac_r}^{\mathrm{RED}}$ has a comparable job-dependent objective to opt. Note that $\mathrm{frac_r}$'s job-dependent objective for jobs in $J_e \cap$ RED is at most,

$$\sum_{j \in J_e \cap \mathrm{RED}} \left(\frac{\delta p_j}{s_{e+1}}\right)^q. \tag{7}$$

Since the fastest group opt can use to process $j \in J_e \cap$ RED is not faster than $G_{e+4\log d+1}$, and its speed is at most $1/d^{4/\gamma}$ times that of G_{e+1} by **Property 3** of smoothed instances, opt's job-dependent objective for jobs in $G_e \cap$ RED is at least

$$\sum_{j \in J_e \cap \text{RED}} \left(\frac{(\delta p_j)/d^2}{s_{e+4\log d+1}} \right)^q,$$

which is greater than its counterpart in opt_r given in (7). (Again, recall that j's size is p_j/d^2 in opt since it processes $J_{\min}$.) Summing over all groups G_e, we have the lemma. □

Lemma 26. $\text{opt}_r^{BLUE} \leq 2^q \text{opt}$.

Proof. Fix a group G_e, $e \geq 4\log d$ (jobs that are fractionally assigned in groups $G_0, G_1, ..., G_{4\log d-1}$ cannot be blue). Consider any infinitesimal job $j \in J_e \cap$ BLUE. As we observed in Section 4, if G_e is the group to which $\text{algo}_{\text{frac}}$ assigns j, then we know that $\alpha_{e-1j} \geq \max\{\alpha_{ej}, \beta_{ej}\}$. This implies that we can bound both frac_r's load-dependent and job-dependent objectives for jobs in $J_e \cap$ BLUE by the load-dependent objective we obtain if all the jobs in $J_e \cap$ BLUE were assigned to G_{e-1}. Hence, frac_r's objective for jobs in $J_e \cap$ BLUE is at most,

$$2|G_{e-1}| \left(\frac{\delta V}{|G_{e-1}| s_{e-1}} \right)^q = \frac{(\delta V)^q}{(S(G_{e-1}))^{q-1}}, \tag{8}$$

where $V := \sum_{j \in J_e \cap \text{BLUE}} p_j$.

We know that opt can only use groups $G_0, G_1, G_2, \ldots, G_{e-4\log d-1}$ to process jobs in $J_e \cap$ BLUE. Let $T := e - 4\log d - 1$. Now we would like to lower bound opt by only considering its load-dependent objective. Thus, we would like to minimize the load-dependent objective when we are asked to process jobs of total size V/d^2 only using groups $G_0, G_1, G_2, ..., G_T$. This is equivalent to the following optimization problem:

$$\min \sum_{t=0}^{T} |G_t| \left(\frac{\delta V_t}{|G_t| s_t} \right)^q = \sum_{t=0}^{T} \frac{(\delta V_t)^q}{(S(G_t))^{q-1}}$$

$$\text{s.t.} \qquad \sum_{t=0}^{T} V_t = V/d^2 \text{ and } V_t \geq 0.$$

Let $S = \sum_{t=0}^{T} S(G_t)$. Note that by **Property 2** of smoothed instances,

$$S = \sum_{t=0}^{T} S(G_t) \leq S(G_{T+1})$$

and

$$\begin{aligned} S(G_{e-1}) &\geq (2^{(e-1)-(T+1)} - 1) \cdot S(G_{T+1}) \\ &\geq (2^{4\log d-1} - 1) \cdot S(G_{T+1}) \geq (d^4/2) \cdot S. \end{aligned}$$

Then, using the convexity of x^q and $q \geq 2$, we have,

$$\begin{aligned} \sum_{t=0}^{T} \frac{(\delta V_t)^q}{(S(G_t))^{q-1}} &= S \sum_{t=0}^{T} \frac{S(G_t)}{S} \left(\frac{\delta V_t}{S(G_t)} \right)^q \\ &\geq S \left(\sum_{t=0}^{T} \frac{S(G_t)}{S} \cdot \frac{\delta V_t}{S(G_t)} \right)^q = \frac{1}{S^{q-1}} \cdot \left(\frac{\delta V}{d^2} \right)^q \\ &\geq \left(\frac{d^4}{2} \right)^{q-1} \cdot \left(\frac{1}{d^2} \right)^q \cdot \frac{(\delta V)^q}{(S(G_{e-1}))^{q-1}} \geq \frac{1}{2^{q-1}} \cdot \left(\frac{\delta V}{d^2} \right)^q, \end{aligned}$$

which is Eqn. (8) times $1/2^q$. Thus, we have the lemma by summing over all G_e, $e \geq 4\log d$. □

Lemmas 24, 25, and 26 collectively imply that $\text{frac}_r \leq O(1)^q \text{opt}$. Recall that to convert frac_r to a valid assignment, we double the amount to which each red and blue job is fractionally assign (which will increase objective h by at most a factor of 2^q). Call this valid fractional solution frac_r'.

Thus, we are left with constructing an integral restricted assignment with objective comparable to that of frac_r'. Consider the solution algo_r which is identical to frac_r, except it fractionally assigns all red and blue jobs based on $\text{algo}_{\text{frac}}$'s solution. We now show the following lemma.

Lemma 27. $\text{algo}_r \leq O(1)^q \text{frac}_r'$.

Proof. Let $\text{opt}_{\max}$ be the optimal integral solution for $J_{\max}$. Define $\text{algo}_r^{\text{RED}}$, $\text{algo}_r^{\text{BLUE}}$, and $\text{algo}_r^{\text{GREY}}$ similarly to how we defined these solutions/objectives for frac_r. Since jobs in $\text{algo}_r^{\text{RED}}$ and $\text{algo}_r^{\text{BLUE}}$ are assigned identically to how they are assigned in our scalar algorithm, we have that

$$\text{algo}_r^{\text{RED}} + \text{algo}_r^{\text{BLUE}} \leq O(1)^q \text{opt}_{\max}.$$

From the definition of grey jobs, we also have $\text{algo}_r^{\text{GREY}} \leq \text{opt}_{\max}$, and since the integrality gap of the problem is constant (e.g., this follows from the results in [3, 13]), we have that $\text{opt}_{\max} \leq O(1)^q \text{frac}_r'$. Finally note that

$$\text{algo}_r \leq O(1)^q (\text{algo}_r^{\text{RED}} + \text{algo}_r^{\text{BLUE}} + \text{algo}_r^{\text{GREY}}).$$

Combining these inequalities, we have the lemma. □

To complete the proof, we observe that applying the scalar rounding algorithm to solution algo_r will produce an integral restricted assignment that will have cost at most $O(1)^q$ times that of algo_r (note that it is a restricted assignment since each red and blue job will be rounded to $G_{m(j)}$). This completes the proof of Lemma 21.

5.2.2 *Proof of Lemma 22.*

Next, we proceed to the second part of the analysis where we show Lemma 22, which states that: *for the restricted assignment problem, our algorithm has a competitive ratio of* $O(\log^2 d)^q$.

Let the optimal restricted assignment be denoted opt_r. For the sake of analysis, we create $O(\log d)$ separate sub-instances, each corresponding to a set of disjoint candidate groups. Namely, let $\mathcal{G}_\ell$ denote the set of jobs j such that $m(j) = G_\ell$ (i.e., the set of jobs whose candidate groups are centered around G_ℓ). This creates $8\log d + 1$ instances, where in the tth instance, $0 \leq t \leq 8\log d$, we schedule jobs with candidate groups $\{\mathcal{G}_t, \mathcal{G}_{t+8\log d+1}, \ldots\}$. It is not hard to verify that each set of candidate groups belongs to a unique instance, and the candidate groups in an instance are disjoint.

First, we argue that the solution produced in each sub-instance is $O(\log d)^q$-competitive against opt_r.

Lemma 28. *Fix a sub-instance S from Phase 2. The objective of the solution produced by the algorithm for S is at most $O(\log d)^q$ times that of the optimal restricted assignment* opt_r.

Proof. First, fix a set of candidate groups $\mathcal{G}$ in S, and consider the solution produced by the VSANY-U algorithm given in [27] for S. This algorithm is $O(\log d + \log m)$-competitive, where m is the number of machines. In our setting, the number of meta machines is

$m = |\mathcal{G}| = O(\log d)$, and thus this algorithm will produce a solution such that the constraints in the makespan minimization problem are violated up to a $O(\log d + \log\log d) = O(\log d)$ factor. Thus it follows that this solution (denote it $\text{algo}_{\mathcal{G}}$) with respect to objective h is at most:

$$\begin{aligned}\text{algo}_{\mathcal{G}} &= |\mathcal{G}| \cdot d \cdot (O(\log d) \cdot \text{opt}_{\mathcal{G}}^{1/q})^q + |\mathcal{G}| \cdot O(\log d) \cdot \text{opt}_{\mathcal{G}} \\ &= O(\log d)^q \cdot \text{opt}_{\mathcal{G}},\end{aligned}$$

since $q \geq \log d$.

Next, observe that since the candidate groups within a sub-instance are disjoint, we have that the algorithm's overall objective in the sub-instance (denote this algo_S) equals $\sum_{\mathcal{G}\in S} \text{algo}_{\mathcal{G}}$. Also, again since candidate groups are disjoint, we have $\sum_{\mathcal{G}\in S} \text{opt}_{\mathcal{G}} \leq \text{opt}_{\text{r}}$. Thus it follows that

$$\text{algo}_S = \sum_{\mathcal{G}\in S} \text{algo}_{\mathcal{G}} = \sum_{\mathcal{G}\in S} O(\log d)^q \cdot \text{opt}_{\mathcal{G}} \leq O(\log d)^q \text{opt}_r. \quad \square$$

Using Lemma 28, we can show that combining the solutions over all sub-instances produces a solution whose objective is at most $O(\log^2 d)^q \cdot \text{opt}_{\text{r}}$ (we relegate a formal proof to the full-version due to space considerations). Thus, we have shown Lemma 22.

ACKNOWLEDGEMENT

We thank Janardhan Kulkarni for many enlightening discussions in the early stages of this work. S. Im and M. Shadloo were supported in part by NSF Awards CCF-1409130 and CCF-1617653. N. Kell and D. Panigrahi were supported in part by NSF Awards CCF-1527084 and CCF-1535972, and by an NSF CAREER Award CCF-1750140.

REFERENCES

[1] Susanne Albers. 1999. Better Bounds for Online Scheduling. *SIAM J. Comput.* 29, 2 (1999), 459–473.
[2] Susanne Albers. 2010. Energy-efficient algorithms. *Commun. ACM* 53, 5 (2010), 86–96.
[3] James Aspnes, Yossi Azar, Amos Fiat, Serge A. Plotkin, and Orli Waarts. 1997. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM* 44, 3 (1997), 486–504.
[4] Adi Avidor, Yossi Azar, and Jiri Sgall. 2001. Ancient and New Algorithms for Load Balancing in the l_p Norm. *Algorithmica* 29, 3 (2001), 422–441.
[5] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. 1995. Load Balancing in the L_p Norm. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995, Milwaukee, Wisconsin, 23-25 October 1995,*. 383–391.
[6] Yossi Azar. 1996. On-line Load Balancing. In *Online Algorithms, The State of the Art.* 178–195.
[7] Yossi Azar, Ilan Reuven Cohen, Amos Fiat, and Alan Roytman. 2016. 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.* 1511–1525.
[8] Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and F. Bruce Shepherd. 2013. Tight bounds for online vector bin packing. In *Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013.* 961–970.
[9] Yossi Azar and Amir Epstein. 2005. Convex programming for scheduling unrelated parallel machines. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005.* 331–337.
[10] Yossi Azar, Joseph Naor, and Raphael Rom. 1995. The Competitiveness of On-Line Assignments. *J. Algorithms* 18, 2 (1995), 221–237.
[11] Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. 1995. New Algorithms for an Ancient Scheduling Problem. *J. Comput. Syst. Sci.* 51, 3 (1995), 359–366.
[12] Yair Bartal, Howard J. Karloff, and Yuval Rabani. 1994. A Better Lower Bound for On-Line Scheduling. *Inf. Process. Lett.* 50, 3 (1994), 113–116.
[13] Piotr Berman, Moses Charikar, and Marek Karpinski. 2000. On-Line Load Balancing for Related Machines. *J. Algorithms* 35, 1 (2000), 108–121.
[14] Ioannis Caragiannis. 2008. Better bounds for online load balancing on unrelated machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008.* 972–981.
[15] Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. 2011. Tight Bounds for Selfish and Greedy Load Balancing. *Algorithmica* 61, 3 (2011), 606–637.
[16] Ashok K. Chandra and C. K. Wong. 1975. Worst-Case Analysis of a Placement Algorithm Related to Storage Allocation. *SIAM J. Comput.* 4, 3 (1975), 249–263.
[17] Chandra Chekuri and Sanjeev Khanna. 2004. On Multidimensional Packing Problems. *SIAM J. Comput.* 33, 4 (2004), 837–851.
[18] R. A. Cody and Edward G. Coffman Jr. 1976. Record Allocation for Minimizing Expected Retrieval Costs on Drum-Like Storage Devices. *J. ACM* 23, 1 (1976), 103–115.
[19] Richard Cole, Vasilis Gkatzelis, and Gagan Goel. 2013. 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.* 251–268.
[20] Ulrich Faigle, Walter Kern, and György Turán. 1989. On the performance of on-line algorithms for partition problems. *Acta Cybern.* 9, 2 (1989), 107–119.
[21] Rudolf Fleischer and Michaela Wahl. 2000. Online Scheduling Revisited. In *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000.* 202–210.
[22] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI).*
[23] Todd Gormley, Nick Reingold, Eric Torng, and Jeffery Westbrook. 2000. 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.* 564–565.
[24] R. L. Graham. 1966. Bounds for certain multiprocessing anomalies. *Siam Journal on Applied Mathematics* (1966).
[25] David G. Harris and Aravind Srinivasan. 2013. The Moser-Tardos Framework with Partial Resampling. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA.* 469–478.
[26] J. F. Rudin III. 2001. Improved Bound for the On-line Scheduling Problem. *PhD thesis, The University of Texas at Dallas* (2001).
[27] Sungjin Im, Nathaniel Kell, Janardhan Kulkarni, and Debmalya Panigrahi. 2015. 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.* 525–544.
[28] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. 2014. Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014.* 313–322.
[29] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. 2015. 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.* 506–524.
[30] David R. Karger, Steven J. Phillips, and Eric Torng. 1996. A Better Algorithm for an Ancient Scheduling Problem. *J. Algorithms* 20, 2 (1996), 400–430.
[31] Gunho Lee, Byung-Gon Chun, and Randy H Katz. 2011. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, Vol. 11.
[32] Adam Meyerson, Alan Roytman, and Brian Tagiku. 2013. 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.* 287–302.
[33] Steven Pelley, David Meisner, Thomas F Wenisch, and James W VanGilder. 2009. Understanding and abstracting total data center power. In *Workshop on Energy-Efficient Design.*
[34] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. 2012. FairCloud: sharing the network in cloud computing. In *ACM SIGCOMM.* ACM, 187–198.
[35] Kirk Pruhs, Jiri Sgall, and Eric Torng. 2004. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis* (2004), 15–1.
[36] Jiri Sgall. 1996. On-line Scheduling. In *Online Algorithms.* 196–231.
[37] Frances Yao, Alan Demers, and Scott Shenker. 1995. A scheduling model for reduced CPU energy. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995, Milwaukee, Wisconsin, 23-25 October 1995.* IEEE, 374–382.