Online Load Balancing for Related Machines

Sungjin Im[*]   Nathaniel Kell[†]   Debmalya Panigrahi[‡]   Maryam Shadloo[§]
UC Merced   Duke University   Duke University   UC Merced

Abstract

In the *load balancing* (or *job scheduling*) problem, introduced by Graham in the 1960s (SIAM J. of Appl. Math. 1966, 1969), jobs arriving online have to be assigned to machines so to minimize an objective defined on machine loads. A long line of work has addressed this problem for both the makespan norm and arbitrary ℓ_q-norms of machine loads. Recent literature (e.g., Azar *et al.*, STOC 2013; Im *et al.*, FOCS 2015) has further expanded the scope of this problem to *vector* loads, to capture jobs with multi-dimensional resource requirements in applications such as data centers. In this paper, we completely resolve the job scheduling problem for both scalar and vector jobs on *related* machines, i.e., where each machine has a given speed and the time taken to process a job is inversely proportional to the speed of the machine it is assigned on. We show the following results:

- *Scalar scheduling.* We give a constant competitive algorithm for optimizing any ℓ_q-norm for (scalar) scheduling on related machines. The only previously known result was for the makespan norm.

- *Vector scheduling.* There are two natural variants for vector scheduling, depending on whether the speed of a machine is dimension-dependent or not. We show a sharp contrast between these two variants, proving that they are respectively equivalent to unrelated machines and identical machines for the makespan norm. We also extend these results to arbitrary ℓ_q-norms of the machine loads. No previous results were known for vector scheduling of related machines.

A key component of our algorithms is a new tool that we call *machine smoothing*, where we replace an arbitrary instance with a *smoothed instance* of the problem. The structural properties of the smoothed instance make it much simpler to argue about various norms of machine loads. We hope that this generic technique will find more applications in other scheduling problems as well.

1   Introduction

The *load balancing* (or *job scheduling*) problem, introduced in the seminal work of Graham in the 1960s [19, 20], asks for an online assignment of jobs to machines so as to minimize some objective defined on machine loads. A long line of work has addressed this problem for both the makespan norm (maximum load) and for other ℓ_q-norms of machine loads (e.g., [9, 26, 2, 16, 2, 15, 10, 18, 22, 3, 8, 11, **?**, 4, 12]). In this paper, we study this problem in the *related* machines setting, where the processing time of a job on a machine is inversely proportional to the speed of the machine. The only previous result for this problem on related machines was a constant-competitive algorithm for the makespan (maximum load) objective [11]. However, in many situations, other ℓ_q-norms of machine loads are more relevant: e.g., the 2-norm is suitable for disk storage [**?**, **?**], whereas q between 2 and 3 is used for modeling energy consumption [**?**, **?**, **?**]. This led to constant-competitive algorithms for arbitrary ℓ_q-norms of machine loads for the special case of identical machines (all machine speeds are equal) [**?**], and to $O(q)$-competitive algorithms for the more general unrelated machines setting (processing times are arbitrary) [4, 12]. But, this problem has remained open for related machines.

Moreover, 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* [13, 7, 28, 25]. This problem is very relevant to 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 [17, 29, 27, 14, 24, 25]. Recently, Im *et al.* [23] 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 a vector job on any dimension of a machine is inversely proportional to the machine speed. But, vector scheduling for related machines had not been addressed previously, either for the makespan norm or for arbitrary ℓ_q norms.

We completely resolve these two sets of problems for scalar and vector scheduling on related machines in this paper. Our first result is for the scalar setting, and gives *a constant-competitive algorithm for optimizing any ℓ_q-norm of machine loads on related machines.* In previous work, the constant competitive ratio for makespan on related machines was obtained by the so-called *slowest-fit* algorithm [11]. The main idea in this algorithm is to guess the optimal makespan, and assign a job arriving online to the slowest machine that can accommodate it without exceeding the optimal makespan by a constant factor. But, this strategy fails for other ℓ_q-norms. Even if we were to guess the optimal value of the norm, this does not tell us the relative contributions of the different machines to the optimal objective. Therefore, guessing the optimal value is not sufficient to fix bounds on the loads of individual machines (unlike makespan, where the guessed optimum gives a bound for the load on each machine). This rules out an assignment strategy like slowest-fit. Instead, we develop a new tool that we call *machine smoothing*, and use it in all our algorithms. Before describing this idea, let us turn to vector scheduling and describe our results for this problem.

Our next contribution in this paper is to *resolve the online vector scheduling problem for related machines*. We show that if machine speeds are dimension-independent (we call this the *homogeneous* case), then the competitive ratio asymptotically matches that of identical machines for the makespan norm. We also extend this result to arbitrary ℓ_q-norms. On the other hand, we show that if machine speeds are dimension-dependent (we call this the *heterogeneous* case), then the competitive ratio asymptotically matches that of unrelated machines. Both homogeneous and heterogeneous speeds are relevant to the practical context and

1

respectively represent situations where clusters only differ in the number of machines or in machine types as well.[1] Unfortunately, the slowest-fit algorithm does not work for vector scheduling on homogeneous machines, even for the makespan norm (see Appendix A for a counterexample). As with scalar scheduling, we again resort to the machine smoothing idea that we describe next.

From a technical perspective, 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 employs an identical machines algorithm within each machine group. But, how do we figure out an assignment of jobs to machine groups? The number of machines in each group might be completely arbitrary, making such assignment a challenging problem. 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. To simultaneously satisfy these goals, we define a machine smoothing procedure that initially groups machines to satisfy the second condition, but then replaces the machines of non-uniform speeds in a group by a suitably defined *equivalent* set of identical machines. We show that this generic transformation can be performed for any given instance, and for any ℓ_q-norm, while only sacrificing a constant factor in the competitive ratio of the algorithm. We call this transformed instance a *smoothed instance* of the problem.

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 gradient descent 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 gradient descent 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 gradient descent 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. Interestingly, while identical machines admit algorithms that optimize all norms *simultaneously* [23], we rule this out for homogeneous related machines (Appendix B). Therefore, our algorithms for vector scheduling for arbitrary ℓ_q-norms use the value of q in the algorithm itself, and this is necessary given our lower bound on optimizing all norms simultaneously.

For the heterogeneous setting, a simple adaptation of the unrelated machines lower bound of $\Omega(\log m)$ gives an instance with $d = \Omega(m)$. This is not interesting because a dependence on $\log d$ is required even for identical machines. Instead, we design an encoding scheme that uses only $d = O(\log m)$ but still manages to show a lower bound of $\Omega(\log m)$. The makespan lower bound for heterogeneous related machines extends to other norms as well, thereby matching known bounds for unrelated machines for all ℓ_q-norms.

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 ar-

---

[1]Note that by scaling, it is sufficient in the homogeneous case for the speeds on different resources to be proportional – they do not need to be exactly equal.

rival. 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 algorithm with a competitive ratio of $O(\log d / \log \log d)$. This is asymptotically tight since it matches a known lower bound for identical machines [23].*

2. *(Section 8) 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 [28, 7, 23].*

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 (we often call this just the q-norm, for brevity):

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

We obtain the following result.

Theorem 2. *For online (scalar) scheduling on related machine for minimizing ℓ_q-norms:*

1. *(Section 4 and Section 5) 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 [2, 15, 10, 18, 22].*

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 6 and Section 7) 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 [23].*

2. *(Section 8) 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 [23].*

3

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.

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 [5, 31, 30] for other results.

The online job scheduling problem was introduced by Graham [19], 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 [9, 26, 2, 16], while the best lowerbound is 1.880 [2, 15, 10, 18, 22]. For the related machines setting, the slowest-fit algorithm is 2-competitive [11], but for unrelated machines, the optimal competitive ratio is $\Theta(\log m)$ [8, 3]. This problem was generalized to arbitrary q-norms by [?] for identical machines and [4, 12] 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 [11].

The multidimensional version of this problem was introduced by Chekuri and Khanna in the offline model [13], who gave a PTAS for constant d. For unrelated machines, they showed a constant lower bound, and the best known approximation factor is $O(\log d / \log \log d)$ due to Harris and Srinivasan [21]. In the online setting, Azar *et al.* [6] and Meyerson *et al.* [28] gave $O(\log d)$-competitive algorithms for identical machines. Recently, Im *et al.* [23] 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.

Roadmap. In the next section, we present the idea of machine smoothing that is a generic tool we use in all the algorithms. This is essentially sufficient for minimizing makespan in vector scheduling on homogeneous machines (Section 3), but we need more ideas for minimizing arbitrary q-norms. Most of these new ideas are for the fractional algorithms, which we present in Sections 4 and 6 for scalar and vector scheduling respectively. The corresponding rounding algorithms are presented in Sections 5 and 7, respectively. Finally, in Section 8, we present our lower bounds for vector scheduling on heterogeneous 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. The goal of this section is to show that such modification is valid, i.e., it does not significantly change the optimal objective.

We will describe the machine smoothing procedure for an arbitrary q-norm objective. First, we articulate the properties that we demand at the end of the transformation.

Definition 1. We say that machines in an instance are *smoothed* if they can be partitioned into groups, $G_0, G_1, G_2, G_3, \cdots$ such that:

- **Property 1**: All machines in each group have *equal* speed.
- **Property 2**: $S(G_l) := \sum_{i \in G_l} s_i^\gamma \geq S(G_0) + S(G_1) + ... + S(G_{l-1})$, where $\gamma = q/(q-1)$.
- **Property 3**: For any two groups G_l and $G_{l'}$ where $l < l'$, any machine in group G_l has a higher speed than any machine in $G_{l'}$ – if two machines have different speeds, their speed differ by at least a factor of 2.

The next lemma claims that any instance can be transformed into a smoothed instance without significantly changing the optimal objective.

Lemma 4. *For any set M of machines (with homogeneous speeds in the case of vector scheduling), we can construct a smoothed set M' of machines such that for any set J of jobs, the respective optimal solutions are related as $\mathrm{opt}(J,M') \leq O(1) \cdot \mathrm{opt}(J,M)$. Furthermore, there exists a mapping $g : M' \to M$ such that if a job scheduled on a machine $i' \in M'$ is scheduled on machine $g(i') \in M$, then the resulting q-norm for the original set M of machines is at most a constant factor larger than the q-norm for the new set M' of machines.*

Proof. We assume (wlog, by scaling) that the fastest machine in M has speed exactly 1. We also round all machine speeds to (negative) powers of $\Gamma := 2^{1/\gamma}$. We order machines in non-increasing order of their speeds, breaking ties arbitrarily. The first group G_0 is the singleton set that has only one machine with speed 1. We now create the remaining groups inductively until every machine is assigned to a group. For $l \geq 1$, exclude machines in $G_0 \cup G_1 \cup ... \cup G_{l-1}$ and define G_l to be the minimal set of the fastest machines i, whose sum of s_i^γ is exactly 2^l. This is always possible to do since we rounded the machine speeds to (negative) powers of Γ, hence s_i^γ are (negative) powers of 2. (The last group G_{L+1} may not satisfy this property.)

Define $S(G) := \sum_{i \in G} s_i^\gamma$ for any group G. For each group G_l, note that $S(G_l) = 2^l$. Let $s_{\min}(G_l)$ denote the lowest speed of all machines in G_l. We replace G_l with a new set G_l' of machines whose speeds are all equal to $s_{\min}(G_l)$, such that $S(G_l') = 2^l$. Let M' denote the machines that we have constructed.

We now prove the first claim that the optimal q-norm increases by at most a constant factor for the new machines M'. Fix an optimal schedule. Since the first group doesn't change, i.e., $G_0 = G_0'$, any job assigned to the machine in G_0 stays there. If the optimal schedule assigns a job j to a machine in G_{l+1}, $1 \leq l \leq L$, we move the job to a machine in G_l'. We let each machine $i' \in G_l'$ process jobs assigned to $T := 2 \cdot \frac{s'^\gamma}{s^\gamma}$ machines $i \in G_{l+1}$, where $s := s_i$ and $s' := s_{i'}$. Note that this is possible since $S(G_l') = |G_l'| \cdot s'^\gamma = 2^l$ and $S(G_{l+1}) = |G_{l+1}| \cdot s^\gamma = 2^{l+1}$, which implies that $|G_l'|/|G_{l+1}| = 2 \cdot \frac{s'^\gamma}{s^\gamma}$. To see that the q-norm increases by a constant factor, consider a fixed dimension and let $u_1, u_2, ..., u_T$ be the volume of jobs assigned to T machines on the fixed dimension. Then, we have

$$\sum_{t=1}^T \left(\frac{u_t}{s}\right)^q \geq T \cdot \left(\frac{\sum_{t=1}^T u_t}{sT}\right)^q = \left(\frac{1}{T}\right)^{q-1} \cdot \left(\frac{s'}{s}\right)^q \cdot \left(\frac{\sum_{t=1}^T u_t}{s'}\right)^q \geq \frac{1}{2^q}\left(\frac{\sum_{t=1}^T u_t}{s'}\right)^q. \tag{1}$$

This implies that the q^q-norm increases by a factor of at most 2^q. 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 4^q, meaning that the optimal q-norm increases by a constant factor.

It now remains to prove the second claim. Consider any online algorithm A. If A assigns a job to a machine $i' \in G_l'$, we assign it to a machine i in G_l; we do not use any machine in G_{L+1}. Fix a group G_l'. We associate each machine with speed s in G_l with $T' := \frac{s^\gamma}{s'^\gamma}$ unique machines in $G_{l'}$ (all these machines have speed s'). This is possible since $S(G_l) = S(G_l')$. Now, using a calculation identical to Eq. (1), we can conclude that the q-norm increases by at most a constant factor in this reassignment.

Also, note that the initial rounding of speeds is only by a constant factor, and hence this also changes the q-norm only by a constant factor. As a consequence, we can now claim that the two properties of the lemma are satisfied by the transformed set of machines M'.

Finally, we are left to prove that the set of machines M' comprise a smoothed instance. It is straightforward to see that these machines, grouped in $G_0', G_1', G_2', \cdots, G_L'$, satisfy the first two properties of smoothed instances. For the third property, we first merge all groups with the same speed. This does not affect the first two properties, and satisfies a weaker version of **Property 3** where machine speeds differ by at least a factor of $2^{1/\gamma}$. To improve this separation to a factor of 2, we merge groups with speeds s' satisfying

$2^l \leq s' < 2^{l+1}$ for each (non-positive) value of i. We now satisfy **Property 2** and **3**, but not **Property 1**. To satisfy **Property 1** as well, we replace the machines of a group G'_l with speeds $2^l \leq s' < 2^{l+1}$ by a new group G''_l containing machines of speed 2^l such that $\sum_{i' \in G'_l} s_{i'}^\gamma = |G''_l| \cdot (2^l)^\gamma$. By mapping machines exactly as above (we omit details for brevity), we can bound the change in the q-norm for both the algorithm and an optimal solution by a constant factor. It is easy to verify that the set of machine groups defined by G'' satisfy all the properties of a smoothed instance. $\qquad\square$

We say that a group is lower than the other group if machines in the group have a lower speed. Note that the set of machines is given to the algorithm a priori. Hence we can find M' and the mapping g offline, and using the mapping g from M' to M, we can convert an online algorithm for the smoothed instance into an online algorithm for the original instance. For this reason, we can assume wlog that machines are smoothed. Also, note that for the makespan norm, the above grouping works exactly as described by setting $\gamma = 1$.

3 Vector Scheduling: Minimizing Makespan

In this section, we give our $O\left(\frac{\log d}{\log \log d}\right)$-competitive algorithm for makespan minimization on homogeneous related machines (the first part of Theorem 1). Recall that in this setting, machine i has a uniform speed vector $\langle s_i, s_i, \cdots, s_i \rangle$, where we refer to s_i as machine i's speed. By scaling, we assume w.l.o.g that the highest speed of any machine is exactly 1. We assume throughout that we have a smoothed instance, which is wlog by Lemma 4.

Algorithm. Since all machines in the same group have equal speed, we use s_l to denote the speed of any machine in group G_l. For simplicity, we say that group G_l's speed is s_l. We assume wlog that we know the value of the optimal makepsan, opt within a constant factor by using a standard doubling technique. We say that a group G_l is permissible for job j if $\max_k \frac{p_j(k)}{s_l} \leq$ opt. The algorithm has two components:

- Assigning jobs to groups of machines: Assign job j to a permissible group G_l with the largest index l; note that G_l has the lowest speed among all permissible groups for job j. Let J_l denote jobs assigned to group G_l.
- Assigning jobs to machines within each group: For each group G_l, run the deterministic $O(\log d / \log \log d)$-competitive algorithm for identical machines in [23] for minimizing makespan to schedule jobs in J_l on machines in G_l.

We formally state the lower bound used in the analysis of the algorithm in [23] used above.

Theorem 5 ([23]). *Suppose that jobs arrive to be scheduled on m identical machines. For any T such that $\max_{k,j} p_j(k) \leq T$ and $\max_k \sum_j p_j(k)/m \leq T$, then there is a deterministic algorithm that yields a schedule with makespan $O\left(\frac{\log d}{\log \log d}\right) \cdot T$.*

The competitive ratio of the algorithm is derived based on two obvious lower bounds, the maximum job size over all dimensions and the average load vectors over m machines. We note that the theorem is stated under the assumption that T is known to the algorithm a priori, but we can again easily remove this assumption by using a standard doubling technique.

We are now ready to complete the proof. Consider any fixed l. Since we schedule jobs J_l on identical machines in G_l, it suffices to show that $\max_{k,j \in J_l} \frac{p_j(k)}{s_l} \leq O(1) \cdot$ opt and $\max_k \frac{\sum_{j \in J_l} p_j(k)}{S(G_l)} \leq O(1) \cdot$ opt. Note that group G_l is permissible for any job in J_l. Hence we have $\max_{k,j \in J_l} \frac{p_j(k)}{s_l} \leq$ opt. Since the optimal scheduler

can schedule jobs in J_l only on machines in $G_1 \cup G_2 \cup \ldots \cup G_l$ (i.e., G_l is the slowest permissible group for jobs in J_l), we have for any dimension k,

$$\sum_{j \in J_l} p_j(k) \leq \sum_{l'=0}^{l} S(G_{l'}) \cdot \text{opt} = \left(S(G_l) + \sum_{l'=0}^{l-1} S(G_{l'}) \right) \cdot \text{opt} \leq S(G_l) \cdot (2 \cdot \text{opt}) \quad \text{(by \textbf{Property 2} of smoothed instances).}$$

Thus, by Theorem 5, the makespan of machines G_l is $O\left(\frac{\log d}{\log \log d} \right) \cdot \text{opt}$.

4 Scalar Scheduling: Minimizing q-norms

As discussed earlier, our algorithm has two parts: a fractional algorithm that assigns jobs fractionally to machines, and a rounding algorithm that converts the fractional solution to an integer solution. We present the fractional algorithm here, and defer the rounding algorithm to Section 5. We will assume throughout that we are working on a smoothed instance, which is wlog by Lemma 4.

To define the fractional algorithm, we first define a fractional relaxation of the q-norm objective. Let us use G to index machine groups; let $|G|$ be the number of machines in group G, p_{Gj} be the processing time of job j on any machine of group G, and x_{Gj} be the fraction of job j assigned to group G. Also, let s_G denote the speed of machines in group G. The (fractional) load of a machine group G is the ratio of the total time for processing the fractional jobs assigned to the group and the number of machines in the group:

$$\Lambda_G = \sum_{j=1}^{n} \frac{1}{|G|} \cdot x_{Gj} p_{Gj}, \quad \text{where } p_{Gj} = \frac{p_j}{s_G}.$$

Then, the fractional objective is:

$$h(x) := \sum_G |G| \cdot (\Lambda_G)^q + \sum_G \sum_j (p_{Gj})^q \cdot x_{Gj}. \tag{2}$$

The first term in $h(x)$ is simply the q^q-norm defined on the fractional loads, and the second term ensures that large jobs do not create a large integrality gap. We call these $f(x) := \sum_G |G| (\Lambda_G)^q$ the *load-dependent* objective, $g(x) := \sum_G \sum_j (p_{Gj})^q \cdot x_{Gj}$ the *job-dependent* objective, and their sum $h(x)$ the *total* objective of solution x.

The goal of the fractional algorithm is to obtain a fractional solution x that is c^q-competitive, for some constant c, for the total objective $h(x)$.

Algorithm. We use a (slightly modified) gradient descent algorithm defined for the objective $h(x)$. To define the algorithm, we denote the two terms in the derivative $\frac{dh(x)}{dx_{Gj}}$ by:

$$\alpha_{Gj} := \frac{df(x)}{dx_{Gj}} = |G| \cdot q \cdot (\Lambda_G)^{q-1} \cdot \frac{1}{|G|} \cdot p_{Gj} = q \cdot (\Lambda_G)^{q-1} \cdot \frac{p_j}{s_G}$$

$$\beta_{Gj} := \frac{dg(x)}{dx_{Gj}} = (p_{Gj})^q = \left(\frac{p_j}{s_G} \right)^q$$

The algorithm assigns an infinitesimal fraction of the current job j to the machine group G that has the minimum value of $\eta_{Gj} := \max(\alpha_{Gj}, \beta_{Gj})$. In case of a tie, the following rule is used:

- If there is a tied machine group with $\alpha_{Gj} < \beta_{Gj}$, then this machine group is used for the assignment. Note that there can only be at most one machine group with this property, by **Property 3** of smoothed instances.

7

- If $\alpha_{Gj} \geq \beta_{Gj}$ for all tied machine groups, then we divide the infinitesimal job among the tied groups in proportion to $|G| \cdot s_G^{\gamma}$, where $\gamma = q/(q-1)$. These proportions are chosen to preserve the condition that the values of α_{Gj} remain tied. This is formally stated in Claim 6, which can be verified by a simple calculation that we defer to the appendix for brevity.

Claim 6. *If a job j is assigned in proportion to $|G| \cdot s_G^{\gamma}$ among machine groups G with identical values of α_{Gj}, where $\gamma = q/(q-1)$, then the value of α_{Gj} remains equal for these machine groups after the assignment.*

Analysis. Our first lemma shows that at any point of time, the values of α_{Gj} for any job j varies monotonically with the speed of the machine groups.

Lemma 7. *At any point of time, if $s_G > s_{G'}$, then $\alpha_{Gj} \geq \alpha_{G'j}$ for any job j.*

Proof. First, note that the lemma holds for all jobs if it does for any single job. We now prove the lemma by showing that it inductively holds for the current job j at any time. For the property to be violated by the current fractional assignment, this assignment must be on group G' with $\alpha_{Gj} = \alpha_{G'j}$. Now, note that $\beta_{G'j} > \beta_{Gj}$ by **Property 3** of smoothed instances. Therefore, the algorithm can make an assignment on G' only if G and G' are tied with

$$\eta_{Gj} = \alpha_{Gj} = \alpha_{G'j} = \eta_{G'j}.$$

In this case, the algorithm assigns job j to groups G and G' in proportion to $|G| \cdot s_G^{\gamma}$ and $|G'| \cdot s_{G'}^{\gamma}$, where $\gamma = q/(q-1)$. This assignment preserves $\alpha_{Gj} = \alpha_{G'j}$ by Claim 6, hence the lemma continues to hold. □

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}. Let $\text{opt}(j)$ (resp., $\text{algo}(j)$) be the machine group on which a job j is assigned by opt (resp., algo). We call the assignment of a fractional job 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 $\text{opt}(j) = \text{algo}(j) = G$, we call it a red assignment if $\beta_{Gj} \geq \alpha_{Gj}$ 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 the job is assigned to the slowest tied machine group. Since $\eta_{Gj} = \alpha_{Gj}$ 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.

We first bound the contribution from red assignments.

Lemma 8. *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 red assignment of job j. 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:

$$\eta_{\text{opt}(j)j} > \eta_{\text{algo}(j)j} = \max(\alpha_{\text{algo}(j)j}, \beta_{\text{algo}(j)j}) \geq \alpha_{\text{algo}(j)j} \geq \alpha_{\text{opt}(j)j} \text{ (by Lemma 7)}.$$

Therefore, $\beta_{\text{opt}(j)j} > \alpha_{\text{algo}(j)j}$. But, since $\beta_{\text{opt}(j)j} > \beta_{\text{algo}(j)j}$ as well, it follows that

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

8

Next, suppose $\mathsf{opt}(j) = \mathsf{algo}(j)$. In this case,

$$\alpha_{\mathsf{algo}(j)j} + \beta_{\mathsf{algo}(j)j} \leq 2\max(\alpha_{\mathsf{algo}(j)j}, \beta_{\mathsf{algo}(j)j}) = 2\beta_{\mathsf{algo}(j)j} = 2\beta_{\mathsf{opt}(j)j},$$

where the second to last equality follows from the definition of red assignments. To complete the proof of the lemma, we note that the increases in $g(x_{\mathsf{opt}})$ are additive across all jobs. $\square$

We are left to bound the total increase in $h(x_{\mathsf{algo}})$ due to blue assignments. For blue assignments, opt assigns the fractional jobs to faster machine groups. To understand the intuition behind our analysis of blue assignments, let us imagine an idealized scenario where algo equalized the values of α_{Gj} across all machine groups G for all jobs j. In this case, algo produced an optimal assignment for the load-dependent objective. Therefore, $f(x_{\mathsf{algo}}) \leq f(x_{\mathsf{opt}})$. The same argument works even if α_{Gj} is not equal for all groups, provided all jobs are blue, by replacing uniformity of α_{Gj} by the monotonicity property from Lemma 7. However, there are two main difficulties with generalizing this argument further. First, for a blue assignment of job j to machine group $\mathsf{algo}(j)$, it may be the case that $\beta_{\mathsf{algo}(j)j} > \alpha_{\mathsf{algo}(j)j}$. In this case, bounding the the load-dependent objective of algo is not sufficient. Second, we need to account for the fact that not all assignments are blue, and the monotonicity guaranteed by Lemma 7 might be contingent on red assignments.

To address the first issue, we specifically consider the blue assignments with $\beta_{\mathsf{algo}(j)j} > \alpha_{\mathsf{algo}(j)j}$; 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 $\mathsf{algo}(j)^+$) that is immediately faster than $\mathsf{algo}(j)$. The idea behind this addition is that $\alpha_{\mathsf{algo}(j)^+j} \geq \eta_{\mathsf{algo}(j)j}$ irrespective of which of $\beta_{\mathsf{algo}(j)j}$ or $\alpha_{\mathsf{algo}(j)j}$ defines $\eta_{\mathsf{algo}(j)j}$. Therefore, we can bound the increase in total objective due to special assignments by the increase in the load-dependent objective 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 $\mathsf{opt}(j)$. Note that for special blue assignments, we have the strict inequality $s_{\mathsf{opt}(j)} > s_{\mathsf{algo}(j)}$; else, we would call it a red assignment. Hence, these additional dummy assignments are also blue assignments.

We now show that these modifications do not significantly change the objectives of the respective solutions, while allowing us to only focus on the load-dependent objectives $f(x_{\mathsf{opt}'})$ and $f(x_{\mathsf{algo}'})$. The first lemma is immediate.

Lemma 9. *The load-dependent objective $f(x_{\mathsf{opt}'})$ in opt' is at most 2^q times the corresponding objective $f(x_{\mathsf{opt}})$ in opt.*

Lemma 10. *The total objective $h(x_{\mathsf{algo}})$ due to blue assignments in algo is at most twice the load-dependent objective $f(x_{\mathsf{algo}'})$ due to blue assignments in algo' .*

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

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

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

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

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

Next, to handle our second issue, we modify opt' to opt'' by adding the load due to red assignments in algo on each machine. 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 as algo. Again, we establish that this transformation does not significantly change the load-dependent objective of opt'.

Lemma 11. *The load-dependent objective $f(x_{\mathsf{opt}''})$ in* opt'' *is at most 2^q times the load-dependent objective $f(x_{\mathsf{opt}'})$ in* opt' *plus 2^{q+1} times the job-dependent objective $g(x_{\mathsf{opt}})$ 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 8, the load-dependent objective in opt'' is at most $2 \cdot 2^q$ times the job-dependent objective $g(x_{\mathsf{opt}})$ in opt. $\qquad\square$

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 theorem on load profiles, which formalizes our earlier intuition.

Lemma 12. *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: $\sum_{G \in \mathcal{G}} \psi_G \cdot |G| \cdot s_G \geq \sum_{G \in \mathcal{G}} \xi_G \cdot |G| \cdot s_G$.*

2. *(Second condition) There exists a $\mu \leq 1$ such that for any two machine groups G and G', we have:*

$$\frac{\xi_G^{q-1}}{s_G} \geq \mu \cdot \frac{\xi_{G'}^{q-1}}{s_{G'}}.$$

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, We set χ_G so that it satisfies

$$\frac{\chi_G^{q-1}}{s_G} = \min_{G':s_{G'} \geq s_G} \frac{\xi_{G'}^{q-1}}{s_{G'}}.$$

Since $\chi_G \leq \xi_G$ for any machine group G, 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,

$$\frac{\chi_G^{q-1}}{s_G} = \min_{G':s_{G'} \geq s_G} \frac{\xi_{G'}^{q-1}}{s_{G'}} \geq \mu \cdot \frac{\xi_G^{q-1}}{s_G}. \tag{3}$$

Now, we use an exchange argument to transform ψ without increasing its load-dependent objective until for every machine group G, we have $\psi_G \geq \chi_G$. In each step of the exchange, we identify the slowest machine group G where $\psi_G < \chi_G$. By the first condition, there must be a machine group G' with $s_{G'} > s_G$ such that

10

$\psi_{G'} > \chi_{G'}$ and for every prefix $\mathcal{G}$ of machine groups in decreasing order of speeds containing G' but not containing G, the following strict inequality holds:

$$\sum_{G \in \mathcal{G}} \psi_G \cdot |G| \cdot s_G > \sum_{G \in \mathcal{G}} \chi_G \cdot |G| \cdot s_G. \tag{4}$$

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

$$\frac{\psi_{G'}^{q-1}}{s_{G'}} > \frac{\chi_{G'}^{q-1}}{s_{G'}} \geq \frac{\chi_G^{q-1}}{s_G} > \frac{\psi_G^{q-1}}{s_G}. \tag{5}$$

Now, we move an infinitesimal job volume from group G' to group G 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 Eqs. (4) and (5). Such moves are repeatedly performed to obtain a load profile ψ'_G with at most the load-dependent objective of ψ, but additionally satisfying $\psi'_G \geq \chi_G$ for all machine groups G.

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 Eq. (3) implies that $\chi_G \geq \mu^{1/(q-1)} \cdot \xi_G$ for every machine group G. $\qquad\square$

We now apply Lemma 12 to algo$'$ and opt$''$ to get our desired bound.

Lemma 13. *The load-dependent objective of* algo$'$ *is at most* 2^q *times the load-dependent objective of* opt$''$.

Proof. In Lemma 12, we set ψ to the load profile of opt$''$ and ξ to the load profile of algo$'$.

The first condition of Lemma 12 follows from the following observations: (a) for blue assignments in algo, $s_{\mathsf{opt}(j)} \geq s_{\mathsf{algo}(j)}$; (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 $s_{\mathsf{opt}(j)} > s_{\mathsf{algo}(j)}$, i.e., $s_{\mathsf{opt}(j)} \geq s_{\mathsf{algo}(j)^+}$.

We now check the second condition of Lemma 12. From Lemma 7, the condition holds with $\mu = 1$ for algo. In algo$'$, the load Λ_{G^+} on a machine group G increases by the total load due to special assignments on machine group G, i.e., by at most $\Lambda_G \cdot \frac{s_G}{s_{G^+}} \leq \Lambda_G$. But, by Lemma 7, $\Lambda_G \leq \Lambda_{G^+}$. Therefore, the load on machine group G^+ increases by at most a factor of 2. It follows that the second condition of Lemma 12 holds with $\mu = 1/2^{q-1}$.

Now, the lemma follows by applying Lemma 12. $\qquad\square$

Combining Lemmas 9, 10, 11, and 13, we obtain the desired bound for blue assignments:

Lemma 14. *The total increase in objective due to blue assignments in* algo *is at most* a^q *times the load-dependent objective of* opt, *for some constant a.*

Lemmas 14 and 8 imply that the algorithm is c^q-competitive on objective $h(x)$ for some constant c, as desired.

5 Scalar Scheduling: Minimizing q-norms (Rounding)

We presented the fractional algorithm for scalar scheduling for q-norms in Section 4. In this section we give a rounding procedure that converts a fractional assignment to an integral assignment with a loss of c^q for some constant c. This result in conjunction with the fractional algorithm from Section 4 implies a $(c \cdot b)^q$-competitive algorithm for optimizing the following objective.

$$h(x) := \sum_i \left(\sum_j x_{ij} p_{ij} \right)^q + \sum_{i,j} (p_{ij})^q x_{ij}. \tag{6}$$

11

Rounding Algorithm. Recall we can assume that machines have been smoothed wlog. It is straightforward to see that we can assume wlog that all machines in each group have identical fractional assignments of jobs. Since all machines in the same group are identical, we can focus on assignments at the granule of groups. In this spirit, we denote the fractional assignment of jobs to groups by $x_{G_l j} := \sum_{i \in G_l} x_{ij}$. Let $m(j)$, which we call j's middle point, be the slowest group G_l (as before, a group's speed is defined as that of any machine in the group) such that j is processed by more than half on machines in groups $G_0, G_1, ..., G_l$, i.e. $\sum_{l \leq m(j)} x_{G_l j} \geq 1/2$; note that $\sum_{l \geq m(j)} x_{G_l j} \geq 1/2$. Then, we 'commit' job j to group G_l. Jobs committed to group G_l are then scheduled greedily within the group (assigned to the machine with the smallest load).

Analysis. We show that committing job j to its middle point group $G_{m(j)}$ and then using greedy algorithm to schedule the job within group $G_{m(j)}$, we only lose $O(1)^q$ factor w.r.t the objective.

Consider any fractional solution x^o. Let $G_{m(j)}$ be the middle point group of job j in x^o. Let's say that a solution/assignment is restricted if each job j must be assigned to groups $G_0, G_1, \ldots, G_{m(j)}$. At a high-level, we first show that this restriction can increase the objective by $O(1)^q$ factor. We then show that the further restriction that job j can only go to machines in $G_{m(j)}$ can increase the objective by $O(1)^q$ factor. Let x' denote a fractional assignment that is obtained from x^o by doubling each job j's assignment to groups $G_0, G_1, ..., G_{m(j)}$ (and discarding some assignments so that $\sum_i x'_{ij} = 1$), and x'' be a fractional assignment where each job j is equally assigned to machines in $G_{m(j)}$.

Lemma 15. $h(x'') \leq O(1)^q h(x^o)$

For a formal proof, we decompose the objective.

$$h_1(x) := \sum_i \left(\sum_j x_{ij} p_{ij} \right)^q \qquad\qquad h_2(x) := \sum_{i,j} p_{ij}^q x_{ij}$$

Lemma 16. $h_1(x'') \leq 2^q h_1(x') \leq 4^q h_1(x^o)$.

Proof. Let J_m denote the set of jobs with the same middle point m. If we only need to schedule jobs J_m, due to the optimality condition (see Claim 6), we can see that $\sum_i (\sum_{j \in J_m} p_{ij} x_{ij})^q$ is minimized when for each $j \in J_m$, x_{ij} is in proportional to s_i^γ for all machines i in groups $G_0, G_1, ..., G_m$. Thus, when $x_{G_0 j}/S(G_0) = x_{G_1 j}/S(G_1) = ... = x_{G_m j}/S(G_m)$, where $x_{G_t j} := \sum_{i \in G_t} x_{ij}$, as before. Knowing that $S(G_m) \geq S(G_0) + S(G_1) + S(G_2) + ... + S(G_{m-1})$ by (at most) doubling the assignments to G_m, we can fully assign jobs in J_m to (machines in) G_m. This will only increase the objective by a factor of 2^q. Further, no two jobs with different middle points are assigned to the same group. This proves the first inequality. The second inequality follows since each machine's load at most doubles when we convert x^o into x'. $\square$

Lemma 17. $h_2(x'') \leq 2 h_2(x^o)$.

Proof. Fix a job j. Any machine $i \in G_{j(m)}$ is faster than any machine i' in $G_{j(m)} \cup G_{j(m)+1} \cup \cdots$. Thus, $p_{ij} \leq p_{i'j}$, hence we can charge j's contribution to the second term in x'' to j's contribution to the second term in x^o on machines in $G_{j(m)} \cup G_{j(m)+1} \cup \cdots$. The factor 2 follows since j is assigned to machines in $G_{j(m)} \cup G_{j(m)+1} \cup \cdots$ by at least half. $\square$

To complete the analysis, it suffices to show that the integral solution $\bar{x}$ produced by the greedy algorithm is c^q-competitive against $h(x'')$ for some constant c.

Lemma 18. $h(\bar{x}) \leq (2^q + 1) h(x'')$

Proof. Fix a group G_l, and let $h_l(x)$ be the objective for just group l. 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., $\Lambda'_i = \Lambda_i - \widehat{p}_i$). Let algo$(j)$ be the machine to which j is assigned by the greedy algorithm. Observe that

$$
\begin{aligned}
h_l(\overline{x}) &= \sum_{i \in G_l} \left(\sum_j p_{ij} \overline{x_{ij}} \right)^q + \sum_{i \in G_l} \sum_j \overline{x_{ij}} p_{ij}^q \\
&= \sum_{i \in G_l} \left(\Lambda'_i + \widehat{p}_i \right)^q + \sum_j p_{\text{algo}(j)j}^q \\
&\leq \sum_{i \in G_l} \left(2\max(\Lambda'_i, \widehat{p}_i) \right)^q + \sum_j p_{\text{algo}(j)j}^q \\
&\leq 2^q \sum_{i \in G_l} \left((\Lambda'_i)^q + \widehat{p_i}^q \right) + \sum_j p_{\text{algo}(j)j}^q \\
&\leq (2^q + 1) \left(\sum_{i \in G_l} (\Lambda'_i)^q + \sum_j p_{\text{algo}(j)j}^q \right) \leq (2^q + 1) h_l(x'').
\end{aligned}
$$

The last inequality follows since x'' assigns all jobs within a group evenly (i.e. $x''_{ij} = 1/|G_l|$ for all i in the group); therefore, since the algorithm assigns greedily, $\sum_{i \in G_l} (\Lambda'_i)^q$ is bounded by $\sum_{i \in G_l} \left(\sum_j x''_{ij} p_{ij} \right)^q$. Similarly, $\sum_j p_{\text{algo}(j)j}^q$ is is equal to $\sum_{i \in G_l} \sum_j x''_{ij} p_{ij}^q$ since all machines have identical speeds within the group.

Summing the bound over all groups l, we obtain that $h(x'') \leq (2^q + 1) h(\overline{x})$. $\square$

6 Vector Scheduling: Minimizing q-norms

As in the previous section on scalar scheduling, we present our fractional algorithm for vector scheduling here, and defer the rounding algorithm to Section 7. In this section we will obtain a fractional solution that is $O(\log^2 d)$-competitive. Then, using the rounding algorithm in Section 7, we will round it with a loss of $O(\log d / \log \log d)$ factor in the competitive ratio, thus proving the first part of Theorem 3. We assume that $q \geq \log d$ since otherwise we can use the any-norm-minimization algorithm for unrelated machines in [23] to find a $O(\log d + q)$-competitive solution. We further assume that $q > 1$ since if $q = 1$, assigning all jobs to the fastest machines yields an optimal solution.

6.1 Overview of Algorithm and Analysis

In this section, our goal will be to find a fractional solution that is $O(\log^2 d)^q$ competitive against the following objective:

$$
\sum_i \sum_k \left(\sum_j p_{ij}(k) x_{ij} \right)^q + \sum_{i,j,k} \left(p_{ij}(k) x_{ij} \right)^q, \tag{7}
$$

where $p_{ij}(k)$ denotes $p_j(k)/s_i$. We first argue that this objective is valid, i.e., if the algorithm is competitive on this relaxation then the algorithm is competitive for our original objective of minimizing the maximum q-norm across all dimensions.

Lemma 19. *An algorithm that is $O(\gamma)^q$-competitive with respect to objective (7) (which sums over all dimensions) implies the algorithm is $O(\gamma)$-competitive for our desired objective stated in the introduction (optimizing for the maximum q-norm across all dimensions; call this the original objective).*

Proof. Recall our definitions of load-dependent, job-dependent, and total objective from Section 4. Let $\|\Lambda^*(k)\|_q$ denote the q-norm of the kth dimension in the optimal solution. Clearly the optimal total objective in a fixed dimension k is within a $O(1)^q$ factor of $\|\Lambda^*(k)\|_q^q$ (since the job-dependent objective is a lower bound on $\|\Lambda^*(k)\|_q^q$). We also have that the optimal solution to objective (7) is at most d times $\|\Lambda^*(k')\|_q^q$, where k' is the dimension with the maximum q-norm. However, since we assume that $q \geq \log d$, we have that $d \leq 2^q$. Thus, putting these observations together, we have that optimal solution to (7) is at most $O(1)^q$ times the optimal solution to the original objective, implying the a $O(\gamma)^q$ competitive algorithm for this relaxation is $O(\gamma)$-competitive on the original objective. $\qquad\square$

As before, we also preprocess machines to create a smoothed instance, which is wlog by Lemma 4. Thus our the objective we will use is the following:

$$\sum_k \sum_G |G| \left(\frac{1}{|G|} \sum_j p_{Gj}(k) x_{Gj} \right)^q + \sum_G \sum_j \left(\sum_k (p_{Gj}(k))^q \right) x_{Gj}, \tag{8}$$

where x_{Gj} denotes that fraction of job j assigned to group G. Recall that within a given group G, we can assume that all jobs assigned to G are spread evenly among the machines in G.

To simplify our presentation, we will assume that each job only has an infinitesimal fraction that needs assigned; namely, we will assume that job j is fully assigned when $\sum_i x_{ij} = \delta$ for an infinitesimally small value $\delta > 0$. This modification can be done by replacing each job j by a set of jobs $j_1, j_2, ..., j_{1/\delta}$ with vector entries $\delta p_j(k)$ for each dimension k and requiring that $\sum_i x_{ij_r} = \delta$ for these newly created jobs. Note that this alternate view does not change the objective considered by the algorithm or how the algorithm works since the algorithm is already making a fractional assignment.

We are now ready to present our algorithm. At a high level, the algorithm assigns each job in two phases. In the first phase, we define a single scalar load derived from the job's maximum load entry and assign it using the scalar algorithm for q norms given in Section 4. This produces a fractional assignment which we will call the *scalar solution*. Using the scalar solution, we then determine a set of *candidate groups* $\mathcal{G}_j$ to which job j can go to in the second phase, i.e.,we only consider assignments where each job j can only go to a group in $\mathcal{G}_j$; call such assignments *restricted assignments*. A key Lemma, which we prove in Section 6.2, is the following:

Lemma 20. *The optimal fractional restricted assignment is at most $O(1)^q$ times the optimal assignment with respect to objective* (8).

Thus, in the second phase, we produce an fractional (vector) assignment that is $O(\log^2 d)^q$-competitive against the optimal restricted assignment, which by Lemma 20 gives us an assignment with the desired competitive ratio. We now describe these two phases in more detail.

Phase 1: Producing the scalar assignment. Let $p_{j,\max} := \max_k p_j(k)$. To define our scalar instance, we set scalar size of job j to be $p_{j,\max}/d^2$. Thus to schedule jobs in this phase, we simply use the algorithm for scalar loads from Section 4.

Let $G_{f(j)}$ be the slowest group where j is assigned in the scalar solution, and let M be the number of groups. Define:

$$\mathcal{G}_j := \{G_{\max\{0, f(j)-4\log d\}}, G_{\max\{0, f(j)-4\log d\}+1}, ..., G_{\min\{M, f(j)+4\log d\}}\},$$

which we call the candidate groups of job j. In other words, $\mathcal{G}_j$ is a collection of $O(\log d)$ consecutive groups containing $G_{f(j)}$ along with (potentially) some slower and some faster groups. Later in Lemma 20,

we will show that there is a $O(1)^q$-approximate assignment w.r.t. (8) where each job j is only assigned to groups in $\mathscr{G}_j$.

Phase 2: Producing the restricted assignment. In this phase, we produce a restricted assignment assignment that is $O(\log^2 d)^q$-competitive against the optimal restricted assignment opt_r, which by Lemma 20 implies a $O(\log^2 d)^q$-competitive solution against the actual optimal solution. To do this, we maintain $O(\log d)$ separate sub-instances, each one corresponding to a set of disjoint candidate groups. Namely, let $\mathscr{G}_G$ denote the set of jobs j such that $f(j) = G$ (i.e., the set of jobs whose candidate groups are centered around G). There will $8\log d + 1$ instances $0, \ldots, 8\log d$, where in the tth instance, we schedule jobs with candidate groups $\{\mathscr{G}_t, \mathscr{G}_{t+8\log d+1}, \mathscr{G}_{t+16\log d+2}, \ldots\}$. It is not hard to verify that each set of candidate groups belongs to a unique instance, and the set of candidate groups within an instance are disjoint.

Within each sub-instance, we will schedule jobs with the same candidate groups separately. Namely, fix a set of candidate groups $\mathscr{G}$ and let $\mathrm{opt}_{\mathscr{G}}$ be the optimal solution (and value of the optimal solution) with respect to objective (8) for scheduling just jobs with candidate groups $\mathscr{G}$. Our goal will be to find a solution that satisfies the following set of constraints:

$$\max_k \max_{G \in \mathscr{G}} \sum_j \frac{1}{|G|} p_{Gj}(k) x_{Gj} \leq \mathrm{opt}_{\mathscr{G}}^{1/q} \text{ and} \tag{9}$$

$$\max_{G \in \mathscr{G}} \sum_j \left(\sum_k (p_{Gj}(k))^q \right) x_{Gj} \leq \mathrm{opt}_{\mathscr{G}}$$

Note that $\mathrm{opt}_{\mathscr{G}}$ satisfies these conditions. Also note that we will assume that $\mathrm{opt}_{\mathscr{G}}$ is known from the outset of the instance (this assumption can be removed by using a standard doubling technique where the algorithm maintains a guess for $\mathrm{opt}_{\mathscr{G}}$ and updates the guess by a factor of 2^q every time it is wrong; however for simplicity, we will assume $\mathrm{opt}_{\mathscr{G}}$ is known for each set of candidate groups $\mathscr{G}$).

We interpret this online problem as the makespan minimization for unrelated machines, i.e., we think of each group G as a meta machine and of each job j as having an averaged load $\frac{\delta p_{Gj}(k)}{|G|}$ on a meta-machine G on dimension k. We also create a special dimension 0 to encode the second set of constraints, where job j has load $\sum_k (\delta p_{Gj}(k))^q$ on meta-machine G on dimension 0. Then, the problem is now reduced to finding an assignment where the makespan on dimension 0 is upper bounded by $\mathrm{opt}_{\mathscr{G}}$, and the makespan on other dimensions from 1 to d is upper bounded by $\mathrm{opt}_{\mathscr{G}}^{1/q}$. In [23], this problem was studied under the name of any norm minimization for unrelated machines (VSANY-U). Using the algorithm in [23], one can find a solution minimizing the $\log(O(|\mathscr{G}|)$-norm on each dimension with the target values $\mathrm{opt}_{\mathscr{G}}^{1/q}$ on dimensions $1, 2, 3, \ldots d$, and $\mathrm{opt}_{\mathscr{G}}$ on dimension 0, which is equivalent to the makespan optimization problem defined by (9) up to a constant factor.

This completes the description of the algorithm for Phase 2. We now show that the Phase 2 assignment is $O(\log^2 d)^q$-competitive ainst the optimal restricted assignment opt_r. First we argue that the solution produced in each sub-instance is $O(\log d)^q$-competitive against opt_r.

Lemma 21. *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 $\mathscr{G}$ in S, and consider the solution produced by the VSANY-U algorithm given in [23] 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 $m = |\mathscr{G}| = O(\log d)$, and thus this algorithm will

15

produce a solution such that the constraints in (9) are violated up to a $O(\log d + \log\log d) = O(\log d)$ factor. Thus is follows that this solution (denote it $\mathsf{algo}_{\mathscr{G}}$) with respect to objective (8) is at most:

$$\mathsf{algo}_{\mathscr{G}} = |\mathscr{G}| \cdot d \cdot (O(\log d) \cdot \mathsf{opt}_{\mathscr{G}}^{1/q})^q + |\mathscr{G}| \cdot O(\log d) \cdot \mathsf{opt}_{\mathscr{G}} = O(\log d)^q \cdot \mathsf{opt}_{\mathscr{G}},$$

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 $\sum_{\mathscr{G} \in S} \mathsf{algo}_{\mathscr{G}}$. Also, again since candidate groups are disjoint, we have $\sum_{\mathscr{G} \in S} \mathsf{opt}_{\mathscr{G}} \leq \mathsf{opt}_r$. Thus is follows that

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

$\square$

Finally, we argue that the overall solution algo (i.e., combining the solutions produced over all sub-instances) is at most $O(\log^2 d)^q \cdot \mathsf{opt}_r$.

Lemma 22. *The solution produced by Phase 2 is at most $O(\log^2 d)^q$ times the optimal restricted assignment.*

Proof. Let $T = O(\log d)$ denote the number of sub-instances. The overall objective that sums over all sub-instances S can be bounded as follows:

$$\mathsf{algo} = \sum_k \sum_G |G| \left(\sum_S \frac{1}{|G|} \sum_{j \in S} p_{Gj}(k) x_{Gj} \right)^q + \sum_S \sum_{G, j \in S} \left(\sum_k (p_{Gj}(k))^q \right) x_{Gj}$$

$$\leq \sum_k \sum_G |G| \left(T \cdot \max_S \left(\frac{1}{|G|} \sum_{j \in S} p_{Gj}(k) x_{Gj} \right) \right)^q + \sum_S \sum_{G, j \in S} \left(\sum_k (p_{Gj}(k))^q \right) x_{Gj}$$

$$\leq T^q \sum_S \sum_k \sum_G |G| \left(\frac{1}{|G|} \sum_{j \in S} p_{Gj}(k) x_{Gj} \right)^q + \sum_S \sum_{G, j \in S} \left(\sum_k (p_{Gj}(k))^q \right) x_{Gj}.$$

$$\leq T^q \sum_S \mathsf{algo}_S \leq T^q \sum_S O(\log d)^q \mathsf{opt}_r = O(\log^2 d)^q \cdot \mathsf{opt}_r,$$

as desired. Note that the the last inequality follows by Lemma 21, and the last equality follows since the are $O(\log d) = O(1)^q$ sub-instances.

$\square$

6.2 Proof of Lemma 20

This section is devoted to showing Lemma 20. Recall that $p_{j,\max} := \max_k p_j(k)$. We first observe that we can assume w.l.o.g. that each job j has size at least $\frac{1}{d^2} p_{j,\max}$ on all dimensions.

Lemma 23. *If we increase each job j's load so that j has load on dimension $\max\{p_j(k), \frac{1}{d^2} p_{j,\max}\}$, objective (8) increases by a factor of at most 2^q.*

Proof. Consider any aggregate load vector on a fixed machine i, $\langle L_1, L_2, ..., L_d \rangle$. Consider an arbitrary dimension, say dimension 1. After the change, L_1 can increase up to $L_1 + \frac{1}{d^2}(L_2 + L_3 + ... + L_d)$. Thus, $(L_1 + \frac{1}{d^2}(L_2 + L_3 + ... + L_d))^q \leq 2^q (L_1)^q + \frac{2^q}{d^q}((L_2)^q + ... + (L_d)^q)$. So one dimension can increase other dimension k's contribution to the objective by only $2^q/d$ times k's contribution before the change. Hence the lemma follows. $\square$

Thus we can assume w.l.o.g. that we run our algorithm after making this change to each job upon arrival. We note that this change is not necessary for the analysis, but it will help simplify our presentation.

Consider an optimal schedule opt and the optimal restricted assignment opt_r. Again to simplify the notation, we let opt and opt_r also denote their objective values, depending on context. We say that a job j is red if it is assigned to a group not in $\mathscr{G}_j$ that is slower than groups in $\mathscr{G}_j$; similarly, the job is said to be blue if it is assigned to a group not in $\mathscr{G}_j$ that is faster than groups in $\mathscr{G}_j$; otherwise, the job is grey. We decompose the objective to analyze the contribution of jobs of each type, separately. In particular, let BLUE, RED, GREY denote set of blue, red, and grey jobs, respectively. Also denote $\text{opt}_r^{\text{BLUE}}$, $\text{opt}_r^{\text{RED}}$, and $\text{opt}_r^{\text{GREY}}$ denote the optimal restricted assignments (and values) that just schedule blue, red, and grey jobs, respectively.

Observe that since grey jobs are scheduled on the same set of machines in both opt and $\text{opt}_r^{\text{GREY}}$, we have that $\text{opt}_r^{\text{GREY}} \leq \text{opt}$. Thus, the following decomposition is immediate.

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

Henceforth, we will focus on bounding opt_r for red and blue jobs. The key idea is to reduce the problem to a single dimensional case. But this reduction is not free – opt_r will have to deal with red and blue jobs of factor d larger sizes than opt. We will still be able to show that opt is considerably large compared to opt_r since opt processes jobs in groups that are so 'out of range.' From now on, we only consider red or blue jobs.

We say that an input is uniform if every job has an equal size over all dimensions. We will consider two uniform inputs derived from the original input. Let J^{unimax} denote the set of jobs where each job j's size vector is replaced with $p_{j,\max} \cdot \langle 1, 1, ..., 1 \rangle$. Similarly, let J^{unimin} denote the set of jobs where each job j's size vector is replaced with $\frac{p_{j,\max}}{d^2} \cdot \langle 1, 1, ..., 1 \rangle$. Note that J^{unimax} is as hard as the original input, and J^{unimin} is as easy as the original input. Since our goal is to upper bound opt_r by opt, we can safely assume that opt_r has to process J^{unimax} while opt does J^{unimin}. Since all jobs have uniform sizes, all dimensions have an equal contribution to the objective. Hence, we can focus on an arbitrary dimension, and ignore all other dimensions. Accordingly, we can now assume that jobs have scalar sizes.

To recap, there are only red or blue jobs. And each job j's size is p_j/d^2 for opt but p_j for opt_r; to simplify the notation we use p_j in place of $p_{j,\max}$. Note that for each job j, opt_r assigns it to groups in $\mathscr{G}_j$, but opt does to other groups. To compare opt_r to opt, we assume that opt_r assigns each job j to group $G_{f(j)}$. Since this is a further restriction to opt_r, we can safely assume. Recall that $G_{f(j)}$ is the group where the single dimensional case algorithm assigns job j with scalar size p_j/d^2. To make our analysis more transparent, for each job j we only keep job j's assignment to $G_{f(j)}$. This is justified since j is assigned to $G_{f(j)}$ by at least half (of its portion δ) as we observed in Section 4. To factor in this, we will lose factor 2^q.

Our remaining goal is to upper bound $\text{opt}_r^{\text{BLUE}}$ and $\text{opt}_r^{\text{RED}}$ by opt. We let J_f denote the set of jobs assigned to G_f in opt_r.

Lemma 25. $\text{opt}_r^{\text{RED}} \leq \text{opt}$.

Proof. Fix a group f. Consider any job $j \in J_f$. The job was assigned to G_f, but not to any slower groups since $\alpha_{G_f j} < \beta_{G_{f+1} j}$. Hence the contribution of red jobs to opt_r's total objective is at most

$$\text{opt}_r(\text{RED}, f) := \delta \sum_{j \in G_f \cap \text{RED}} \left(\frac{p_j}{s_{G_{f+1}}} \right)^q.$$

Knowing the fastest group opt can use to process j is $G_{f+4\log d+1}$, and its speed is at most $1/d^4$ times that of G_{f+1}, opt's job-dependent objective for jobs in $G_f \cap \text{RED}$ is at least $\delta \sum_{j \in G_f \cap \text{RED}} (\frac{p_j/d^2}{s_{G_{f+4\log d+1}}})^q \geq \text{opt}_r(\text{RED}, f)$. Summing over all f, we have the lemma. $\qquad \square$

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

Proof. Fix a group f. Consider blue jobs assigned to G_f in opt_r. As we observed in Section 4, if G_f is the slowest group to which the single dimensional case algorithm assigns j, then we know that $\alpha_{G_{f-1}j} \geq \max\{\alpha_{G_f j}, \beta_{G_f j}\}$. Hence we can upper bound opt_r's total objective for jobs $J_f \cap \text{BLUE}$ by $\mathsf{opt}_r(\text{BLUE}, f) := |G_{f-1}|(\frac{V}{|G_{f-1}|s_{G_{f-1}}})^q = \frac{V^q}{(S(G_{f-1}))^{q-1}}$ where $V := \sum_{j \in J_f \cap \text{BLUE}} p_j$. We know that opt can only use groups G_0, G_1, G_2, ..., $G_{f-4\log d-1}$ to process jobs in $G_f \cap \text{BLUE}$. Let $T := f - 4\log d - 1$. 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're asked to process jobs of total size V/d^2 only using groups $G_0, G_1, G_2, ..., G_T$. In other words, we would like to minimize $\sum_{1 \leq t \leq T} |G_t|(\frac{V_t}{|G_t|s_{G_t}})^q = \sum_{1 \leq t \leq T}(\frac{V_t^q}{(S(G_t))^{q-1}})$ subject to $\sum_{1 \leq t \leq T} V_t = V/d^2$. By an easy algebra, we can see that the minimum is $\frac{V}{d^2}\eta^{q-1}$ where $\eta := \frac{V/d^2}{\sum_{1 \leq t \leq T} S(G_t)} \geq d^2 \cdot \frac{V}{S(G_{m-1})}$. Thus, $\frac{V}{d^2}\eta^{q-1} \geq \mathsf{opt}_r(\text{BLUE}, m)$ due to Properties 2 and 3; recall that q is an integer greater than 1. By summing over all f, we have the lemma. $\square$

Thus we have proven Lemma 20.

7 Vector Scheduling: Minimizing q-norms (Rounding)

In this section we give a rounding procedure that converts a fractional assignment to an integral assignment with a loss of $O(\frac{\log d}{\log\log d})$ factor in the competitive ratio for minimizing the q norm when machines have homogeneous speeds. We will use the following objective, which is equivalent to our original objective up to a constant factor; see Lemma 19.

$$h(x) := \max_k \sum_i \left(\sum_i p_{ij}(k)x_{ij} \right)^q + \sum_{i,j}(p_{ij}(k))^q x_{ij} \tag{10}$$

Rounding Algorithm. Like scalar scheduling, since all machines in the same group are identical we focus on assignment of jobs to groups. We define $x_{G_l j}$ and $m(j)$ as before. We 'commit' job j to the middle point group $G_{m(j)}$. Then, we schedule job j Jobs on one of machines of this group by following the $O(\frac{\log d}{\log\log d})$-competitive algorithm for vector identical machines.

Analysis. We first show that we can commit each job j to its middle point group, $G_{m(j)}$ without losing more than $O(1)^q$ factor w.r.t .(10). We define x^o, x' and x'' the same as previous section.

Lemma 27. $h(x'') \leq O(1)^q h(x^o)$.

To prove this lemma let's decompose the objective. Note that $h(x)$ and $h_1(x) + h_2(x)$ are within factor 2.

$$h_1(x) := \max_k h_{1,k}(x) \qquad \text{where } h_{1,k}(x) := \sum_i(\sum_j x_{ij}p_{ij}(k))^q$$

$$h_2(x) := \max_k h_{2,k}(x) \qquad \text{where } h_{2,k}(x) := \sum_{i,j}(p_{ij}(k))^q x_{ij}$$

Lemma 28. *For any x, $h(x) \leq h_1(x) + h_2(x)$.*

Proof. Immediate from the definition of h, h_1 and h_2. $\square$

18

Lemma 29. $h_1(x'') \leq 2^q h_1(x') \leq 4^q h_1(x^o)$.

Proof. Fix a dimension k. Consider scalar scheduling in this dimension. From the lemma 16, we can say $h_{1,k}(x'') \leq 2^q h_{1,k}(x') \leq 4^q h_{1,k}(x^o)$ for each k. Definition of $h_1(x)$ follows the lemma. $\square$

Lemma 30. $h_2(x'') \leq 2h_2(x^o)$.

Proof. For each k, with the same argument as lemma 17, we have $h_{2,k}(x'') \leq 2h_{2,k}(x^o)$. The lemma follows from definition of h_2. $\square$

From the above lemmas, the desired Lemma 27 follows.

It now remains to show that given a fractional assignment where each job is assigned to only one group consisting of identical machines, we can convert it into an integral assignment online using a $O(\frac{\log d}{\log \log d})$-competitive algorithm for d-dimensional identical machines. Our goal is to establish a competitive ratio of $O((\frac{\log d}{\log \log d})^q)$ against $h(x)$ when all machines are identical. Let m denote the number of machines.

Although [23] gives a $O((\frac{\log d}{\log \log d})^{\frac{q-1}{q}})$-competitive algorithm for the q norm, here we only present an online rounding algorithm that loses a competitive ratio of $O(\frac{\log d}{\log \log d})$. The reason we present a slightly worse competitive ratio is because we need to argue against the objective $h(x)$, hence we can't do some part of the preprocessing done in [23]. Also since we already lose an additional $O(\log^2 d)$ factor in other places, we choose not to further optimize this ratio.

The rounding algorithm we use here is essentially the $O(\log d / \log \log d)$-competitive makespan minimization algorithm for identical machines [23]. Let's use the objective (8). As discussed before, this is equivalent to $h(x)$ up to a constant factor for minimizing the q norm. By a standard doubling trick, we can assume w.l.o.g. that we know the final maximum average load on any dimension, i.e., $A := \max_k \sum_j p_j(k)/m$. Note that the first term in the objective is lower bounded by mA^q; since all machines are identical, we assume w.l.o.g. that the speed is 1. We say that a job j is big on dimensions k if $p_{ij}(k) \geq A$. Our rounding algorithm ensures that every machine gets at most η big jobs on any dimension, and its total load of small jobs is at most ηA where $\eta = O(\log d / \log \log d)$ where an appropriate constant is hidden. This can be done by a independent rounding, followed by a postprocessing that takes are of 'overloaded' jobs. The idea is, using standard concentration inequalities, to show that only a very small fraction of jobs need to be 'reassigned' in the postprocessing. This randomized rounding can be derandomzied using a potential function argument.

To see that this guarantee is sufficient to establish the desired competitiveness w.r.t. the objective (8), consider any fixed machine i. Let J_s and J_b denote small and big jobs assigned to i, respectively. Let X_{ij} denote a binary variable such that $X_{ij} = 1$ if and only if j is assigned to machine i after the rounding. Then, machine i's contribution to the first term in the objective is,

$$\sum_k (\sum_j p_j(k) X_{ij})^q$$
$$\leq 2^q \sum_k (\sum_{j \in J_s} p_j(k) X_{ij})^q + 2^q \sum_k (\sum_{j \in J_b} p_j(k) X_{ij})^q$$
$$\leq 2^q \sum_k (\eta A)^q + 2^q \sum_k \eta^q \sum_{j \in J_b} (p_j(k))^q X_{ij},$$

where the last inequality follows since each machine contains at most η big jobs and each machine has at

19

most ηA load of small jobs on any dimension k. Summing over all machines, we have

$$\sum_i \sum_k (\sum_j p_j(k)X_{ij})^q$$
$$\leq (2\eta)^q dmA^q + (2\eta)^q \sum_{j \in J_b} (p_j(k))^q$$

Since all machines are identical, the second term of the objective (8) is the same for all feasible assignments. Hence we have shown that our final solution is $(4\eta)^q d$-competitive against the optimal fractional solution w.r.t. objective (8). Since $q \geq \log d$, $(4\eta)^q d = O(\log d / \log \log d)^q$, as desired.

8 Heterogeneous Machines

In this section we give our $\Omega(\log m)$ lower bound for related machines with heterogeneous speeds (the second part of Theorem 1), i.e., the speed vector for a fixed machine need not be uniform. This result also extends to a $\Omega(\log d + q)$ lower bound for generic q-norms, thereby showing a $\Omega(\log m + \log d)$ lower bound for the makespan case when $q = \log m$.

We (the adversary) construct our online lower bound instance as follows. Let $d = 2h + 1$ be the number of dimensions; there will be 2^h machines in total. All speeds will be either be 1 or arbitrarily slow; for simplicity, we will just say these machines have speed 0. To define each speed $s_i(k)$, we first pair off $2h$ of the total $2h + 1$ dimensions into h pairs, and order these pairs $1, \ldots, h$ arbitrarily; we will call the remaining dimension that is not paired the *aggregate* dimension; we will call the other dimensions that are paired *pattern* dimensions.

For each pair of pattern dimensions (k, k') and a fixed machine i, we will define machine speeds so that either $s_i(k) = 1$ and $s_i(k') = 0$ or vice versa. We say that (k, k') has *speed pattern A* in the former case and speed pattern B in the latter. To define speeds over all machines in pattern dimensions, we can think of taking the set of all 2^h strings As and Bs of length h, mapping each one to a unique machine, and then using the string to define the corresponding speed pattern. For example, if we map string t to machine i and the ℓth character of t is B, then for the ℓth dimension pair (k, k') we set $s_i(k) = 0$ and $s_i(k') = 1$. Finally, we will simply fix the speed of all machines in the aggregate dimension to be 1. This completes the definition of machine speeds in the instance.

Now we describe the job sequence for the instance. Jobs will be issued in h rounds $1, \ldots, h$, one for each dimension pair. Throughout the instance, we maintain a set of *active* machines in which the algorithm can still use; in other words, jobs will be defined so that they cannot be assigned to inactive machines. Denote the set of active machines at the beginning of round ℓ as T_ℓ. At the start of the instance all machines are active, and then each round, the number of active machines is halved, where the goal is to limit the algorithm to machines that have already been heavily loaded in the aggregated dimension.

The adversary maintains active machines as follows: Suppose we are in the ℓth round of the instance. For this round, we will call a machine an *A machine* if it has speed pattern A in the ℓth dimension pair; B machines are defined similarly, and inductively assume there are an equal amount of A and B machines in T_ℓ. We will issue a set of jobs J_ℓ such that $|J_\ell| = |T_\ell| = m/2^{\ell-1}$, i.e., we issue as many jobs as there are active machines. After the algorithm assigns the jobs in J_ℓ, we then observe which set, the A machines or B machines, has received the majority of the load among machines in T_ℓ in the aggregate dimension up until this point in the instance. We will then define future jobs so that they are limited to this more heavily loaded set of machines. For example, letting (k, k') denote the ℓth dimension pair, if machines in T_ℓ with pattern A have received a majority of the jobs up until this point, then for all future jobs j after this round we define

$p_j(k) = 1$ and $p_j(k') = 0$ so that the algorithm is forced to continue to use these machines. We will call this the *majority speed pattern* for round ℓ. We will also define each job so that it has load 1 in the aggregate dimension, and the loads for dimension pairs $\ell + 1, \ldots, h$ are defined to be 0. This completes the description of the construction, and one can verify that this induction is well defined.

The resulting instance will force a makespan of $h = \Omega(\log m)$ on some machine in the aggregate dimension. This claim is implied by the following lemma:

Lemma 31. *The average load on active machines in the aggregate dimension at the start of round $\ell + 1$ is at least ℓ.*

Proof. Consider the start of round ℓ, and inductively assume the average load on active machines T_ℓ is at least $\ell - 1$. Recall that the number of active machines $|T_\ell| = m/2^{\ell-1}$ at the beginning of this round. Since we issue $m/2^{\ell-1}$ jobs and they can only go to active machines, the average load for T_ℓ machines increases by 1, i.e., it is now at least ℓ. Furthermore, since we pick the majority speed pattern based on which pattern currently has more load in the aggregate dimension, it is not hard to verify that the average for these $m/2^\ell$ machines must also be at least ℓ. Since these machines with the majority speed pattern will be the new active machines for round $\ell + 1$, the proof of the lemma now follows by induction. $\qquad\square$

To complete the argument, observe that it is possible to "reverse" the decisions of the algorithm to get a makespan of at most 2 on all machines and dimensions. In particular, the optimal solution assigns all jobs in the ℓth round to the machines that do not correspond to the majority speed pattern in the ℓth dimension pair (i.e., if the majority speed pattern was A for a round, then all jobs are assigned to B machines, and vice versa). Since in each round half the machines are A and B machines, respectively, and we issue as many jobs as there are active machines, this will produce a load of 2 on the machines that do not correspond to the majority speed pattern. This completes our proof for the second part of Theorem 1.

We now extend the above lower bound to show a lower bound of $\Omega(\log d + q)$ for case when each dimension can be evaluated with arbitrary q-norm for $1 \le q \le \log m$. In the above construction, the load vector in the aggregate dimension at the end of the instance has load vector identical of that in the $\Omega(q)$ lower bound for the single-dimensional unrelated machines lower bound (see [4]), and thus the above construction also gives a lower bound of $\Omega(q)$. To obtain a lower bound of $\Omega(\log d)$, we add m *additional* dimensions $1, \ldots, m$ to the above construction. Note that now $d = \Theta(m)$. The speed in additional dimension i is 1 on machine i and arbitrarily fast on all other machines. These additional dimensions receive the same load that the aggregate does. Based on the construction, there will some additional dimension i' with load $\Omega(\log m)$ on machine i' at the end of the instance (the machine that produces a load of $\Omega(\log m)$ in the aggregate dimension). Note that the optimal solutions obtains a q-norm of $(2^q)^{(1/q)} = O(1)$ on all additional dimensions, whereas the algorithm's solution has produced a q-norm of $((c \log m)^q)^{(1/q)} = \Omega(\log d)$ for additional dimension i'. This completes the extension.

Acknowledgement

References

[1] Faraz Ahmad, Srimat T Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 61–74, 2012.

[2] Susanne Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999.

[3] James Aspnes, Yossi Azar, Amos Fiat, Serge A. Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[4] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Load balancing in the l_p norm. In *FOCS*, pages 383–391, 1995.

[5] Yossi Azar. On-line load balancing. In *Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996)*, pages 178–195, 1996.

[6] Yossi Azar, Ilan Reuven Cohen, Amos Fiat, and Alan Roytman. Packing small vectors. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1511–1525, 2016.

[7] Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and F. Bruce Shepherd. Tight bounds for online vector bin packing. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 961–970, 2013.

[8] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, 1995.

[9] Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.

[10] Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.

[11] Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35(1):108–121, 2000.

[12] Ioannis Caragiannis. Better bounds for online load balancing on unrelated machines. In *SODA*, pages 972–981, 2008.

[13] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004.

[14] Richard Cole, Vasilis Gkatzelis, and Gagan Goel. Mechanism design for fair division: allocating divisible items without payments. In *Proc. 14th ACM conference on Electronic commerce*, pages 251–268, 2013.

[15] Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta Cybern.*, 9(2):107–119, 1989.

[16] Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, pages 202–210, 2000.

[17] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

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

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

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

[21] David G. Harris and Aravind Srinivasan. 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*, pages 469–478, 2013.

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

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

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

[25] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive flow time algorithms for polyhedral scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 506–524, 2015.

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

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

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

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

[30] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis*, pages 15–1, 2004.

[31] Jiri Sgall. On-line scheduling. In *Online Algorithms*, pages 196–231, 1996.

[32] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proc. 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 29–42, 2008.

A   Counterexample for Slowest-Fit in Vector Scheduling

The previously known scalar scheduling algorithm [11] for related machines with the makespan norm only loses a constant factor by using *slowest-fit*: assign a job to the slowest machine that can accommodate it without exceeding the desired competitive ratio. What if we use the same rule for assigning jobs in vector scheduling for the makespan norm? Unfortunately, this strategy fails.

Example: Consider a set of homogeneous related machines where there are 2^g machines of speed $1/2^g$, $g \in \{1, 2, \cdots, d\}$ – let's index this group by g. Let c be the desired competitive ratio or equivalently the maximum average load we allow for each group. Note that all groups have an equal 'processing power,' $2^g \cdot 1/2^g = 1$. Any job released is sufficiently small so that it can be assigned to any group, i.e., $2^d \cdot ||p_j||_\infty \leq 1$ for all j. We release jobs in d phases. In the gth phase, every job has size $1/2^d$ on dimension g, and extremely tiny sizes on the other dimensions. There will arrive $c \cdot 2^d$ such jobs in this phase. In the spirit of slowest-fit, these jobs will be assigned to group g, eventually making the group hit the threshold c on the average load on dimension g. Note that other dimensions are barely used. After all the d phases, from d to 1, we now release tiny jobs with size $1/2^d$ on all dimensions. However, every group has hit the predetermined threshold on a distinct dimension, thus can't accept any more jobs. In contrast, it is easy to see an optimal schedule with makespan c/d (ignoring the extremely tiny sizes).

The problem with slowest-fit is that it excessively preserves fast machines for big jobs that may arrive in the future. In particular, it fails to realize in the above instance that all the groups have exactly the same processing power. This suggests that the slowest-fit strategy would work better if we can ensure that the slower groups have larger processing power, and therefore should receive most of the jobs. We artificially ensure this by grouping machines not by speed, but in a way such that the total processing power of the groups increases exponentially as we move to slower machines. While this creates the desired distribution of processing power, we no longer have the property that the machines in the same group have similar speeds. However, we manage to show that we can replace the (actual) machines in each group by a set of (simulated) identical machines with the same cumulative processing power, but with speed equal to that of the slowest machine in the group, without increasing the optimal makespan by more than a constant factor. This constitutes our machine smoothing technique that is given in Section 2.

B   Impossibility for All Norms Minimization in Vector Scheduling

In this section, we provide an instance that rules out all norms minimization even for related machines with homogeneous speeds. This will distinguish related machines from identical machines, for which a

logarithmic competitive algorithm was shown for all norms minimization [23].

Instance. There are two type of machines, fast and slow. There are t fast machines with speed 1 and t^2 slow machines with speed $1/\sqrt{t}$. The number of dimensions $d = t^2 + 1$. There are t^2 jobs, and each job has size 1 on a distinct dimension and size $1/t$ on a dimension that is shared by all jobs – we call this dimension the common dimension; we call the the other dimensions dummy dimensions.

If we place an arbitrary set of t jobs on each fast machine, the makespan is 1, and the L_4 norm of the loads is 1 on any dummy dimension and $t^{1/4}$ on the common dimension. Now let's see how the makespan norm and L_4 norm change when we assign each job to a distinct slow machine. Note that the makespan is now $\sqrt{t}$. The L_4 norm also increases to $\sqrt{t}$ on any dummy dimension, but decreases to $((1/\sqrt{t})^4 t^2)^{1/4} = 1$ on the common dimension. Thus one can improve some norm on a specific dimension by a factor polynomial in d while sacrificing others.

C   Proof of Claim 6

We recall the claim: *If a job j is assigned in proportion to $|G| \cdot s_G^\gamma$ among machine groups G with identical values of α_{Gj}, where $\gamma = q/(q-1)$, then the value of α_{Gj} remains equal for these machine groups after the assignment.*

Proof. Recall that

$$\alpha_{Gj} := q \cdot (\Lambda_G)^{q-1} \cdot \frac{p_j}{s_G} \tag{11}$$

Therefore its derivative with respect to an assignment x_{ij} is:

$$\frac{d\alpha_{Gj}}{dx_{ij}} = q(q-1) \cdot (\Lambda_G)^{q-2} \cdot \frac{p_j^2}{s_G^2}$$

Substituting for Λ_i using (11) we have:

$$\frac{d\alpha_{Gj}}{dx_{ij}} = q(q-1) \cdot \left(\frac{s_G \alpha_{Gj}}{p_j \cdot q} \right)^{\frac{q-2}{q-1}} \cdot \frac{p_j^2}{s_G^2} \tag{12}$$

To keep α_{Gj} values equal while dividing x_{ij} infinitesimally among the groups, we should assign mass inversely proportional to $\frac{d\alpha_{Gj}}{dx_{ij}}$ times $|G|$ to each group G. However, since all G already have equal α_{Gj} upon the assignment, all terms in $\frac{d\alpha_{Gj}}{dx_{ij}}$ except for S_G are common across these groups. Thus, each group should receive mass in proportion to $S_G^{2-(q-2)/(q-1)}|G| = S_G^\gamma |G|$. $\square$