Improved Bounds for Distributed Load Balancing

Sepehr Assadi
Rutgers University, New Brunswick, NJ, USA
sepehr.assadi@rutgers.edu

Aaron Bernstein
Rutgers University, New Brunswick, NJ, USA
bernstei@gmail.com

Zachary Langley
Rutgers University, New Brunswick, NJ, USA
zach.langley@rutgers.edu

Abstract

In the load balancing problem, the input is an n-vertex bipartite graph $G = (C \cup S, E)$ – where the two sides of the bipartite graph are referred to as the clients and the servers – and a positive weight for each client $c \in C$. The algorithm must assign each client $c \in C$ to an adjacent server $s \in S$. The load of a server is then the weighted sum of all the clients assigned to it. The goal is to compute an assignment that minimizes some function of the server loads, typically either the maximum server load (i.e., the ℓ_∞-norm) or the ℓ_p-norm of the server loads. This problem has a variety of applications and has been widely studied under several different names, including: scheduling with restricted assignment, semi-matching, and distributed backup placement.

We study load balancing in the distributed setting. There are two existing results in the CONGEST model. Czygrinow et al. [DISC 2012] showed a 2-approximation for unweighted clients with round-complexity $O(\Delta^5)$, where Δ is the maximum degree of the input graph. Halldórsson et al. [SPAA 2015] showed an $O(\log n/\log\log n)$-approximation for unweighted clients and $O(\log^2 n/\log\log n)$-approximation for weighted clients with round-complexity polylog(n).

In this paper, we show the first distributed algorithms to compute an $O(1)$-approximation to the load balancing problem in polylog(n) rounds:

- In the CONGEST model, we give an $O(1)$-approximation algorithm in polylog(n) rounds for unweighted clients. For weighted clients, the approximation ratio is $O(\log n)$.
- In the less constrained LOCAL model, we give an $O(1)$-approximation algorithm for weighted clients in polylog(n) rounds.

Our approach also has implications for the standard sequential setting in which we obtain the first $O(1)$-approximation for this problem that runs in near-linear time. A 2-approximation is already known, but it requires solving a linear program and is hence much slower. Finally, we note that all of our results *simultaneously* approximate all ℓ_p-norms, including the ℓ_∞-norm.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Load Balancing, Distributed Algorithms, Matching, Semi-Matching

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.1

Related Version A full version of the paper is available at `https://arxiv.org/abs/2008.04148`.

Funding *Aaron Bernstein*: This work was done while funded by NSF CAREER Grant 1942010 and Simons Collaboration on Algorithms and Geometry.

34th International Symposium on Distributed Computing (DISC 2020).
Editor: Hagit Attiya; Article No. 1; pp. 1:1–1:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this paper, we study the *load balancing* problem. The input is a bipartite graph $G = (C \cup S, E)$, where we refer to the sets C and S the *clients* and *servers*, respectively. The goal is to find an assignment of clients to servers such that no server is assigned too many clients. To be more precise, we define the load of a server in an assignment to be the number of clients assigned to it, and we are interested in finding an assignment that minimizes the maximum load of any server (or minimizes the ℓ_p-norm of the server loads – we will discuss this objective more later in the introduction).

The load balancing problem has a rich history in the scheduling literature as the *job scheduling with restricted assignment problem* [14, 6, 19, 13], in the distributed computing literature as the *backup placement problem* [12, 22, 3], in sensor networks [23, 21] and peer-to-peer systems [18, 26, 25] as the *load-balanced data gathering tree construction problem*, and more generally as a relaxation of the bipartite matching problem known as the *semi-matching problem* [13, 8, 9, 17]. We refer the reader to [13, 9, 12] for more background.

Our primary focus in this paper is on the load balancing problem in a distributed setting, where clients and servers correspond to separate nodes in a network. Communication through the network happens in synchronous rounds, where in each round, every node can send $O(\log n)$ bits to its neighbors over any of its incident edges (formally, we work in the CONGEST model – see Section 2 for more details). In the distributed setting, the load balancing problem generalizes the distributed backup placement problem with replication factor one (introduced in [12]), where the nodes (corresponding to clients) in a distributed network may have memory faults and therefore wish to store backup copies of their data at neighboring nodes (corresponding to servers). Since backup-nodes may incur faults as well, the number of nodes that select the same backup-node should be minimized. See the full version of this paper [1] for the exact formulation of the distributed backup placement problem and for how some of our results extend to the more general version of the problem with arbitrary replication factor.

A simple distributed algorithm for the load balancing problem in which the nodes myopically reassign themselves to a server with smaller load eventually converges to an $O(\frac{\log n}{\log\log n})$-approximation, where n is the number of nodes in the network [10, 16], but as was shown in Halldórsson et al. [12], the algorithm requires $\Omega(\sqrt{n})$ rounds. The same paper [12] shows a way of circumventing this costly process and gives a distributed algorithm that achieves the same $O(\frac{\log n}{\log\log n})$-approximation in only $\mathrm{polylog}(n)$ rounds. On the other hand, Czygrinow et al. [8] show a distributed $O(1)$-approximation (precisely, a 2-approximation) that requires $O(\Delta^5)$ rounds, where Δ is the maximum degree of a node in the network. This algorithm is highly efficient for low-degree networks but is again too expensive for high-degree graphs.

This state-of-affairs is the starting point of our work: *Can we obtain the best of both worlds, namely, an $O(1)$-approximation algorithm in $\mathrm{polylog}(n)$ rounds?*

Our first contribution. Our first main contribution in this paper is an affirmative answer to this question.

▶ **Result 1** (Formalized in Theorem 6)**.** *We give an $O(1)$-approximate randomized distributed algorithm for load balancing in the* CONGEST *model that runs in $O(\log^5 n)$ rounds.*

At the core of our algorithm is a new structural lemma for the load balancing problem. Informally speaking, we show that eliminating all "short augmenting paths" of length $O(\log n)$ is sufficient to assign *all* clients to servers with load a constant factor as much as the optimum (**Lemma 3**). In conjunction with ideas from [12], this effectively reduces the load balancing problem to that of finding a matching with no short augmenting paths, which can be solved using the by-now standard algorithm of Lotker et al. [20].

Our second contribution. Next, we consider the *weighted* load balancing problem in which every client comes with a weight. The load of a server is then the total weight of the clients assigned to it. The goal, as before, is to minimize the maximum load of any server. Halldórsson et al. [12] also studied the weighted problem and gave an $O(\frac{\log^2 n}{\log\log n})$-approximation in polylog(n) rounds using a simple reduction to the unweighted case.

Using the same weighted-to-unweighted reduction, our algorithm in Result 1 also implies an $O(\log n)$-approximation for the weighted load balancing problem in polylog(n) rounds of the CONGEST model. Our main technical contribution in this paper is a new algorithm for this problem that achieves an $O(1)$-approximation in the less constrained LOCAL model, in which communication over edges in each round is unbounded.

▶ **Result 2** (Formalized in Theorem 15). *We give an $O(1)$-approximate randomized distributed algorithm for weighted load balancing in $\log^3 n$ rounds of the* LOCAL *model.*

Our LOCAL algorithm consists of two main components: a distributed algorithm for (approximately) solving a relaxed version of the problem where each client c with weight $w(c)$ should be assigned to $w(c)$ adjacent servers with multiplicity – a *split assignment* – and a novel distributed rounding procedure. Using our structural result in Lemma 3, we can find a split assignment by approximately solving (or rather, eliminating short augmenting paths in) a generalized b-matching problem with *edge capacities*. We are not aware of any efficient algorithm for this problem in the CONGEST model, but we can show that a simple extension of the work of [20] can solve this problem in polylog(n) rounds in the LOCAL model. The rounding step is also based on a new application of our Lemma 3 that allows us to circumvent the typical use of "cycle canceling" procedures for rounding fractional matching LP solutions into integral ones, which do not translate to efficient distributed algorithms.

We now turn to two important extensions of Results 1 and 2. The first is the more general problem of all-norm load balancing, and the second is a fast sequential algorithm.

Approximating all norms. Recall that our goal in the load balancing problem has been to minimize the maximum load of any server. Assuming we denote the loads of servers under some assignment A by a vector $L_A := [L_A(s_1), L_A(s_2), \ldots, L_A(s_n)]$ for all $s_i \in S$, minimizing the maximum server load is equivalent to minimizing $\|L_A\|_\infty$, i.e., the ℓ_∞-norm of L_A. Depending on the application, however, miniziming this norm may not be the most natural notion of a "balanced" assignment; if some server requires vastly more load than the other servers, an ℓ_∞-norm-minimizing assignment may put needlessly large load on those other servers.

As a result, it is natural to consider minimizing some other ℓ_p-norm of L_A for some $p \geq 1$. This is done, for instance, in [13, 9, 17], which considered ℓ_2-norms. An even more general objective is the *all-norm* problem, studied in [2, 5, 7, 13], where the goal is to *simultaneously* optimize with respect to every ℓ_p-norm. These results compute an assignment which is an $O(1)$-approximation (or even optimal) simultaneously with respect to all ℓ_p-norms, including $p = \infty$ (*a priori*, even the existence of such an assignment is not clear).

All of our results extend to the all-norm problem without any increase in approximation factor or round-complexity. In particular, in the CONGEST model, we give randomized distributed $O(1)$- and $O(\log n)$-approximation algorithms for all-norm load balancing in polylog(n) rounds, in the unweighted and weighted variant of the problem, respectively (**Theorem 11**). In the LOCAL model, the approximation ratio for the weighted problem can be reduced to $O(1)$ as well (**Theorem 15**).

Faster sequential algorithms. Finally, we show that our new approach to weighted load balancing can also be used to design a near-linear time algorithm for this problem in the *sequential* setting. We give a deterministic $O(m \log^3(n))$ time algorithm for the $O(1)$-approximate all-norm load balancing problem in the sequential setting (**Theorem 20**).

Previously, a deterministic $O(m\sqrt{n} \log n)$ time for the exact problem in case of unweighted graphs was given in [9]. The weighted variant of the problem is NP-hard [2]; 2-approximate algorithms were shown in [2] and [7], but they are based on solving, respectively, the linear and convex programming relaxations of the problem exactly using the ellipsoid algorithm, and thus are much slower than the algorithm we present.

2 Preliminaries

Notation. For any function $f : A \to \mathbb{N}$ and $B \subseteq A$, we use the notation $f(B) = \sum_{b \in B} f(b)$ to sum f over all elements in B. For any integer $t \geq 1$, we denote $[t] := \{1, \ldots, t\}$.

Throughout, we assume $G = (C \cup S, E)$ is a bipartite graph. We refer to C and S as the *clients* and the *servers*, respectively. We let uv denote the edge between vertices u and v and let $\delta(v)$ denote the set of edges incident to a vertex v. We use n as number of vertices in G and m as the number of edges in G.

Load balancing. In the load balancing problem, the input is a bipartite graph $G = (C \cup S, E)$ together with a client weight function $w : C \to [W]$. The output is an assignment $A : C \to S$ mapping every client to one of its adjacent servers. The *load* $L_A(s)$ of a server $s \in S$ under assignment A is the sum of the weights of the clients assigned to it: $L_A(s) = w(A^{-1}(s))$. The *maximum load* of an assignment A is the maximum load of any server under A. We refer to the problem of computing an assignment of minimum load as the *(weighted) min-max load balancing problem.*

As mentioned in the introduction, the min-max objective can be generalized by considering any ℓ_p-norm of L_A, defined as $\|L_A\|_p = \left(\sum_{s \in S} (L_A(s))^p\right)^{1/p}$. For brevity, we also use the notation $\|A\|_p := \|L_A\|_p$. In the language of norms, the min-max objective corresponds to minimizing the load vector's ℓ_∞-norm. When the goal is to find an assignment A that simultaneously minimizes $\|A\|_p$ for all $p \geq 1$, including $p = \infty$, the problem is called the *(weighted) all-norm load balancing problem.* Prior results in [2, 5, 7, 13] show the *existence* of an assignment that can (approximately) minimize all these norms simultaneously. In particular, we use the following result due to Harvey et al. [13] in our proofs (see also [5]).

▶ **Lemma 1** ([13]). *Given any instance of the unweighted load balancing problem, there exists an assignment A^* that simultaneously minimizes $\|A^*\|_p$ for all $p \geq 1$, including $p = \infty$.*

***b*-matchings.** In addition to assignments, we will also work with *b-matchings.* For a vertex capacity function $b : V \to \mathbb{Z}^+$, a b-matching is an assignment $x : E \to \mathbb{Z}^+$ of integer multiplicities to edges so that for every vertex v, the sum $x(\delta(v))$ of the multiplicities of the edges incident to v does not exceed $b(v)$.

Since we will focus solely on the case when G is bipartite and $V = C \cup S$, it will be convenient to split b into two separate capacity functions, one for the clients and one for the servers. We use $\kappa : C \to \mathbb{Z}^+$ to denote the *client capacities* and $\tau : S \to \mathbb{Z}^+$ to denote the *server capacities.* A (κ, τ)-*matching* is then a function $x : E \to \mathbb{Z}^+$ assigning multiplicities to edges such that

$$\sum_{s \in N(c)} x(cs) \leq \kappa(c) \tag{1}$$

for every client c and

$$\sum_{c \in N(s)} x(cs) \leq \tau(s)$$

for every server s. A (κ, τ)-matching is *client-perfect* if (1) holds with equality for all $c \in C$. We say that a server s (resp. client c) is x-*saturated* if $x(\delta(s)) = \tau(s)$ (resp. $x(\delta(c)) = \kappa(c)$). If a vertex is not x-saturated, then it is x-*unsaturated*. An x-*augmenting path* is a path $v_1, \ldots, v_{2k+1}$ such that v_1 and v_{2k+1} are x-unsaturated and $x(v_{2i+1}v_{2i+2}) > 0$ for all $0 \leq i < k$.

We will make repeated use of the following simple remark.

▶ Remark 2. When all client weights are one (the unweighted case), a client-perfect $(1, \tau)$-matching induces an assignment of maximum load at most $\max_{s \in S} \tau(s)$, and vice versa.

Note that the remark does not generalize to weighted clients; under a (w, τ)-matching, a client may be split across multiple servers, which does not correspond to a proper assignment.

The LOCAL and CONGEST models. In both the LOCAL and the CONGEST models of distributed computation, each vertex of the input graph hosts a processor that initially only knows its neighbors and its weight. Following a standard assumption, we assume that all vertices know n and the maximum weight W. Computation proceeds in synchronous rounds; in each round, vertices may send messages to their neighbors and then receive messages from their neighbors in lockstep. Local computation is free – we are only interested in the *round complexity*, the number of rounds required by the algorithm.

The LOCAL and CONGEST models differ in that in the LOCAL model, vertices can send and receive arbitrarily large messages to and from their neighbors, while in the CONGEST model, the communication between adjacent vertices in each round is capped at $O(\log n)$.

3 A Structural Lemma

A crucial component of our results is a structural observation about approximate (κ, τ)-matchings in the context of the load balancing problem, which is inspired by results from online load balancing [11, 4]: if a graph contains *some* client-perfect (κ, τ)-matching, then *every* $(\kappa, 2\tau)$-matching is either client-perfect or can be augmented via an augmenting path of logarithmic length. Formally, and more generally, we have the following lemma.

▶ **Lemma 3.** *If G contains a client-perfect (κ, τ)-matching and x is a $(\kappa, \alpha\tau)$-matching for $\alpha > 1$, then either x is client-perfect or there is an x-augmenting path of length at most $2\lceil \log_\alpha \tau(S) \rceil + 1$.*

Proof. Suppose G contains a client-perfect (κ, τ)-matching x^*. To simplify the discussion, we define a directed multigraph D on $V(G)$ whose arcs are oriented edges in the support of x and x^* as follows. For every $cs \in E(G)$ with $c \in C$ and $s \in S$, D has $x(cs)$ copies of the arc (s, c) and $x^*(cs)$ copies of the arc (c, s) and no other arcs. Notice that every directed path in D starting at an x-unsaturated client and ending at an x-unsaturated server corresponds to an x-augmenting path in G.

Suppose x is not client-perfect and let $c \in C$ be an x-unsaturated client. Let $k \in \mathbb{N}$ be fixed and define U_k to be the set of vertices reachable via a walk of length k from c in D. Call U_k *full* if u is x-saturated for all $u \in U_k$.

The lemma follows from two simple claims:

1. if U_{2k+1} is not full, then G contains an x-augmenting path of length at most $2k+1$; and
2. if U_{2k+1} is full, then $\tau(U_{2k+3}) \geq \alpha\tau(U_{2k+1})$.

The first claim follows from the fact that a directed walk contains a directed path with the same endpoints and from the correspondence noted earlier between directed paths with unsaturated endpoints in D and augmenting paths in G.

We proceed to the second claim. If $s \in U_{2k+1}$ and U_{2k+1} is full, then s is x-saturated and the out-degree of s is $\alpha\tau(s)$. Thus, the total out-degree of U_{2k+1} – and also the total in-degree of U_{2k+2} – is $\alpha\tau(U_{2k+1})$. Now we use the fact that the out-degree of a client $c \in U_{2k+2}$ is at least as large as its in-degree. This follows simply from the fact that the in-degree must be at most $\kappa(c)$, and since x^* is client-perfect, the out-degree is exactly $\kappa(c)$. Following the arcs once more, the total in-degree of U_{2k+3} is at least $\alpha\tau(U_{2k+1})$. Finally, since the in-degree of U_{2k+3} is also point-wise less than τ, we have $\alpha\tau(U_{2k+1}) \leq \tau(U_{2k+3})$.

Now we show how the two claims together imply the lemma. If any U_{2i+1} for $i \leq \lceil \log_\alpha(\tau(S)) \rceil$ is not full, we are done by the first claim. Otherwise, the sums of capacities grow exponentially starting with $\tau(U_1) \geq 1$. Inductively, $|U_{2i+1}| \geq \alpha^i$ for all $i \in \mathbb{N}$. For $k = \lceil \log_\alpha \tau(S) \rceil$, therefore, we have $\tau(U_{2k+3}) \geq \alpha\tau(S)$, a contradiction. Thus, not all $\{U_{2i+1}\}$ are full. ◀

4 Unweighted Load Balancing

Assuming an algorithm to eliminate augmenting paths up to a certain length efficiently, the structural lemma from the previous section almost immediately implies an algorithm for the unweighted load balancing problem. To obtain an algorithm for eliminating short augmenting paths, we use the following lemma which is implied by Lemma 24 in [12].

▶ **Lemma 4** ([12]). *There exists an $O(k^3 \log n)$-round randomized algorithm in the* CONGEST *model that, with high probability, given a graph $G = (C \cup S, E)$, a positive integer k, and server capacity function τ, computes a $(1, \tau)$-matching with no augmenting paths of length less than k.*

The proof of Lemma 4 combines two existing results. The algorithm of Lotker et al. [20] computes a (1,1)-matching with no augmenting paths of length $\leq k$ in $O(k^3 \log n)$ rounds. Halldórsson et al. [12] then show a black-box extension from (1,1)-matching to (1, τ)-matching which does not increase the round-complexity; see [12] for more details.

▶ Remark 5. Both our algorithm and the algorithm of [12] use the above lemma as a starting point. But the algorithm of [12] only removes short augmenting paths to ensure that the $(1, \tau)$-matching is approximately optimal. Since a near-optimal matching is still not an assignment (it is not client-perfect), they then use a different set of tools to convert an approximate $(1, \tau)$-matching to an $O(\log n / \log\log n)$-approximate assignment.

Our analysis, by contrast, directly exploits the non-existence of short augmenting paths via Lemma 3. We thus avoid the additional conversion of [12], which leads to a better approximation ratio, as well as a simpler algorithm.

Approximating the ℓ_∞-norm (the min-max load balancing problem). Let B^* be the optimum ℓ_∞-norm. We will first describe an algorithm that assumes as input some $B \geq 2B^*$. The algorithm begins by using Lemma 4 to compute a $(1, B)$-matching x with no augmenting paths of length $4\lceil \log_2 n \rceil + 1$. The sum of the server capacities is at most nB, and since clients have unitary weight, we can assume $B \leq n$. Therefore, by Lemma 3, since there are no x-augmenting paths of length $2\lceil \log_2(nB) \rceil + 1 \leq 4\lceil \log_2 n \rceil + 1$, we know that x is necessarily client-perfect. A client c can now assign itself to the vertex it is matched to under x.

To remove the assumption that we are given a $B \geq 2B^*$, we run the algorithm above $\log n$ times with $B = 1, 2, 4, \dots, n$. For every run where $B \geq 2B^*$, the algorithm will successfully assign every client. Note, however, that in the distributed setting, there is no efficient way for the clients to determine the smallest B for which the algorithm successfully matched every client. Instead, each client c *locally* assigns itself according to the run with smallest B that succeeded—i.e., according to the first run in which c was matched. We show that the resulting assignment has maximum load at most $8B^*$. See Algorithm 1 for a concise treatment.

Algorithm 1 Approximate unweighted load balancing in the CONGEST model.

---

for $B \in \{1, 2, 4, \dots, n\}$ **do**
  compute a $(1, B)$-matching x_B with no augmenting paths of length $4\lceil \log n \rceil + 1$
end
each client c locally finds the minimum B such that c is matched in x_B and assigns itself to the server it is matched to in x_B

▶ **Theorem 6.** *In the* CONGEST *model, there is a randomized algorithm (Algorithm 1) that with high probability computes an 8-approximation to the min-max load balancing problem in* $O(\log^5 n)$ *rounds.*

Proof. First observe that since augmenting paths with respect to a $(1, n)$-matching have length at most 1, all clients are assigned in x_n. Therefore the algorithm always outputs an assignment of all clients.

Let B^* be the optimal maximum load and B be the smallest power of 2 that is at least $2B^*$ (we have $B \leq 4B^*$). The $(1, B)$-matching x_B computed in the main loop of Algorithm 1 will be client-perfect by Lemma 3, and so no client will assign itself according to x'_B for $B' > B$. A single server is only assigned at most i clients from x_i, and since $x_1, x_2, x_4, \dots, x_B$ are the only assignments in play, the total load of a server under any combination of these assignments is at most $1 + 2 + 4 + \cdots + B < 2B \leq 8B^*$. Thus every server has load at most $8B^*$.

Finally, each x_B is computed in $O(\log^4 n)$ rounds with high probability by Lemma 4. We compute them sequentially, resulting in total round complexity of $O(\log^5 n)$. ◀

▶ Remark 7. In the LOCAL model, the round complexity of Algorithm 1 is $O(\log^3 n)$. One log factor is shaved off of Lemma 4 because the algorithm of [20] for finding a (1,1)-matching is faster in the LOCAL model. The second log factor is shaved off by running the for-loop of Algorithm 1 in parallel for every B.

Approximating all ℓ_p-norms simultaneously (the all-norm load balancing problem). The assignment produced by Algorithm 1 in fact does more than approximate the optimal ℓ_∞-norm; it also simultaneously approximates every ℓ_p-norm for $p \geq 1$, as we will now show.

Recall that by Lemma 1, when clients are unweighted, there is an all-norm optimal assignment that simultaneously minimizes the ℓ_p-norm for all $p \geq 1$, including $p = \infty$. Let $\mathcal{A}^*$ be the set of all all-norm optimal assignments; by Lemma 1, the set $\mathcal{A}^*$ is non-empty. We need the following key definition.

▶ **Definition 8.** *The* level $\ell(s)$ *of a server s is the maximum load of s over all assignments in $\mathcal{A}^*$, i.e.,* $\ell(s) := \max_{A\in\mathcal{A}^*} L_A(s)$. *The* level $\ell(c)$ *of a client c is* $\ell(c) := \max_{A\in\mathcal{A}^*} L_A(A(c))$.

▶ **Lemma 9.** *For $i \in [n]$, let $C_i \subseteq C$ be the set of clients at level i and let $S_i \subseteq S$ be the set of servers at level i. There are no edges in G from C_i to S_j for any $j < i$.*

Proof. Fix a client $c \in C_i$ and let A be an all-norm optimal assignment such that $L_A(A(c)) = i$. Suppose to obtain a contradiction that $N(c) \cap S_j \neq \emptyset$ and let $s \in N(c) \cap S_j$. Consider the assignment A' which maps c to s and is otherwise identical to A. Comparing the load vector of A to A', one entry of the load vector moves from i to $i-1$, another entry moves from j to $j+1$, and the rest remain unchanged. Since $j < i$, it follows that $\|A'\|_p \leq \|A\|_p$ for all p. If $\|A'\|_p < \|A\|_p$ for some p, then A is not all-norm optimal, a contradiction. Otherwise, the norms are equal for all p. But then there is an all-norm optimal assignment, namely A', in which s has level $j+1$, contradicting that the level of s is j. ◀

▶ **Lemma 10.** *If a client $c \in C$ has level at most ℓ and x is a $(1, B)$-matching with $B \geq 2\ell$ such that there are no x-augmenting paths of length at most $4\lceil \log n \rceil + 1$ in the graph, then c is matched under x.*

Proof. Consider the graph G' constructed by removing all servers of load larger than ℓ from G and let x' be x restricted to G'. Since every augmenting path in G' is an augmenting path in G, there are no x'-augmenting paths of length $\leq 4\lceil \log n \rceil + 1$ in G'. Note that any optimal assignment in G restricted to G' has maximum load at most ℓ. Since x' is a $(1, B)$-matching, G' has a client-perfect $(1, \ell)$-matching, and $B \geq 2\ell$, it follows that x' is client-perfect in G' by Lemma 3. As x' is the restriction of x to G', it follows that x matches all clients of level at most ℓ in G. ◀

▶ **Theorem 11.** *In the* CONGEST *model, there is a randomized algorithm (Algorithm 1) that with high probability computes an 8-approximation to the all-norm load balancing problem in $O(\log^5 n)$ rounds.*

Proof. For each client c, let B_c be the smallest power of two that is at least $2\ell(c)$. By Lemma 10, each client c will be assigned in x_{B_c}. Fix a server s. Lemma 9 implies that the load of s is only determined by clients whose level is at most $\ell(s)$ as no other clients can be adjacent to s. Since each x_B contributes at most B to the load of s and only contributes for $B \leq 4\ell(s)$, we obtain that $L_A(s) \leq 1 + 2 + 4 + \cdots + 4\ell(s) \leq 8\ell(s)$. Thus,

$$\|A\|_p = \left(\sum_{s\in S} (L_A(s))^p\right)^{1/p} \leq \left(\sum_{s\in S} (8\ell(s))^p\right)^{1/p} = 8\,\|A^*\|_p\,. \qquad ◀$$

5 Weighted Load Balancing

In this section, we describe our algorithms for the weighted load balancing problem. We start by showing that with the simple reduction in [12] from unweighted to weighted load balancing, our unweighted algorithm (Algorithm 1) also implies an $O(\log n)$-approximate polylog(n)-round CONGEST algorithm for weighted instances. We then turn to the main result of this section: an $O(1)$-approximate polylog(n)-round LOCAL algorithm for the weighted load balancing problem. We conclude this section with an $O(1)$-approximate sequential algorithm that runs in near-linear time.

As our goals in this section are to obtain, at best, an $O(1)$-approximation, we may assume that all client weights are powers of two. If not, rounding weights up to the nearest power of two will at most double the approximation ratio. We can assume similarly that the maximum weight $W \le n$. Indeed, clients with load less than W/n can collectively distribute at most W weight across the servers and can therefore be assigned arbitrarily. Thus, when $W > n$, clients can simply rescale their own weight by n/W (and round it up to the nearest integer).

Throughout this section, we denote by C_i the set of clients whose weight is exactly 2^i. (By our previous assumption, the sets $\{C_i\}$ partition C.) We let $G_i := G[C_i, N(C_i)]$ be the induced graph on C_i and its neighborhood.

An $O(\log n)$-approximation in the CONGEST model. We begin with an easy corollary of our unweighted algorithm following a simple reduction in [12].

▶ **Theorem 12.** *In the* CONGEST *model, an $O(\log n)$-approximation to the all-norm weighted load balancing problem can be computed with high probability in $O(\log^5 n)$ rounds.*

Proof. Consider the following algorithm: For each weight class i, compute an assignment A_i of G_i using Algorithm 1 by treating all clients as having weight 1. Then, have each client in C_i assign itself according to A_i.

Since all A_i's can be computed in parallel (as the graphs G_i are edge-disjoint, only one of the parallel copies need to communicate over an edge), the algorithm runs in $O(\log^5 n)$ rounds. We now show that the resulting assignment A is $O(\log n)$-approximate for all norms.

Fix any $p \ge 1$ including $p = \infty$; let A^* be an assignment for G with minimum ℓ_p-norm, and let A_i^* be an assignment for G_i with minimum ℓ_p-norm. Clearly $\|A_i^*\|_p \le \|A^*\|_p$ for all i. By Theorem 11, $\|A_i\|_p \le 8\|A_i^*\|_p \le 8\|A^*\|_p$. It follows that

$$\|A\|_p = \|A_1 + \cdots + A_{\log n}\|_p \le \|A_1\|_p + \cdots + \|A_{\log n}\|_p \le 8\log(n)\|A^*\|_p . \qquad ◀$$

Preliminaries for the weighted algorithms. Though they use entirely different techniques, the LOCAL and sequential algorithms of the next two subsections both follow the same high-level approach: first compute a split assignment, then round it into an integral one.

▶ **Definition 13.** *Let $G = (C \cup S, E)$ be a bipartite graph with client weights $w : C \to \mathbb{Z}^+$. A* split assignment *y_f in G is a client-perfect (w, ∞)-matching (so servers have unbounded capacity). For every server s, the* load *$L_{y_f}(s)$ is the sum of edge-multiplicities incident to s.*

Notice that split assignments are a relaxation of standard assignments by allowing clients to be assigned to several different servers at once, contributing an integral load to each server, provided that the total load distributed by the client does not exceed its weight.

We will also need the following notion. Define the *client-expanded graph* $\widetilde{G}$ of G as the graph formed by making $w(c)$ copies of each client c. Formally, for each $c \in C$, the client-expanded graph has vertices $c_1, \ldots, c_{w(c)}$ and an edge between c_i and s for all $i \in [w(c)]$ if and only if G has an edge between c and s.

▶ **Observation 14.** *A split assignment y_f in G corresponds to an integral assignment in the client-expanded graph $\tilde{G}$ with the same server loads. Thus, since $\tilde{G}$ is unweighted, by Lemma 1 there exists an all-norm optimal split assignment y_f^*.*

5.1 An $O(1)$-approximation in the LOCAL model

Our main result in this section is the following theorem.

▶ **Theorem 15.** *In the* LOCAL *model, there is a randomized algorithm (Algorithm 2) that with high probability computes an $O(1)$-approximation to the weighted all-norm load balancing problem in $O(\log^3 n)$ rounds.*

We will need the next rounding lemma to describe our algorithm; the proof is standard.

▶ **Lemma 16.** *If $G = (C \cup S, E)$ contains a client-perfect (κ, τ)-matching x, then there exists an assignment $A : C \to S$ such that for all servers $s \in S$,*

$$L_A(s) \leq \tau(s) + \max_{c \in A^{-1}(s)} \kappa(c).$$

Proof of Lemma 16. Consider the set of edges F in the support of x. If $C \subseteq F$ is a cycle, we can alternately increase and decrease the value of $x(e)$ on each edge e of the cycle by $\min_{f \in C} x(f)$ to break the cycle without changing $x(\delta(v))$ for any $v \in V$ (this cycle can only be of even length as the input graph is bipartite). Thus, we may assume that the support of x has no cycles and thus is a forest.

We can next turn F into a collection of stars centered on servers. This done by rooting each tree T in the support of F arbitrarily, picking each server s which has a client parent-node c, and setting the edge $x(cs) = \kappa(c)$ and $x(cs') = 0$ for all other $s' \in N(c)$. This clearly satisfies the requirement of client c and the load on server s can only ever be increased by $\max_{c \in A^{-1}(s)} \kappa(c)$ as each server can only have one parent client. At this point, in F, any client is assigned to exactly one server and thus we obtain an integral solution in which the load of any server s is at most $\tau(s) + \max_{c \in A^{-1}(s)} \kappa(c)$, finalizing the proof. ◀

Our LOCAL algorithm consists of two main parts, an algorithm for solving the split load balancing problem and a rounding procedure, which we describe now in turn.

Computing a split assignment. The first step of the LOCAL algorithm is to compute an assignment $\tilde{A}$ in the client-expanded graph $\widetilde{G}$ of G using Algorithm 1. Note that in the LOCAL model, each client c can simulate all "new" clients $c_1, \ldots, c_{w(c)}$ in Algorithm 1 without any overhead in the round complexity.[1] As mentioned in Observation 14, the assignment $\tilde{A}$ corresponds to a split assignment with the same server loads. To limit the amount of notation in the algorithm description, we will sometimes refer to $\tilde{A}$ as a split assignment in G, although formally it is an assignment in $\tilde{G}$.

The guarantees of Algorithm 1 tell us that $\tilde{A}$ has small ℓ_p-norm. The next step is to use $\tilde{A}$ to find an integral assignment without much loss in the norm.

A "rounding" procedure. We would now ideally round the split assignment $\tilde{A}$ into an integral assignment, but even in the LOCAL model we cannot afford to run such a procedure directly. The fact that a good rounding exists, however, is enough for us to apply Lemma 3 to obtain a similarly good assignment, as we show below.

For each i, let $\tilde{A}_i$ be $\tilde{A}$ restricted to G_i. Lemma 16 states that there is a way to round $\tilde{A}_i$ into an assignment with load $\tau_i(s) = L^i_{\tilde{A}_i}(s) + 2^i$ for servers s assigned to by $\tilde{A}_i$ and $\tau_i(s) = 0$ for the remaining servers. Treating the clients as unweighted, $\tilde{A}_i$ corresponds to

---

[1] We remark that computing this assignment is the only step of our weighted algorithm that does not run efficiently in the CONGEST model, precisely because this simulation not possible in the CONGEST model in polylog(n) rounds.

Algorithm 2 Approximate weighted (all-norm) load balancing in the LOCAL model.

1 emulate Algorithm 1 on $\widetilde{G}$ to compute an assignment $\tilde{A}$
2 **for** $i \in \{1, 2, 4, \ldots, n\}$ *in parallel* **do**
3   let $\tilde{A}_i$ be $\tilde{A}$ restricted to $\widetilde{G}_i$
4   let $\tau_i(s) = \begin{cases} L_{\tilde{A}_i}(s) + 2^i, & \text{if } L_{\tilde{A}_i}(s) > 0 \\ 0, & \text{otherwise} \end{cases}$
    ▷ *by Lemma 16, an assignment with load vector point-wise less than τ_i exists in G_i*
    ▷ *therefore, scaling clients in G_i to weight 1, a $(1, \lceil 2^{-i}\tau_i \rceil)$-matching exists*
5   treating G_i as unweighted, compute a $(1, 2\lceil 2^{-i}\tau_i \rceil)$-matching x_i in G_i with no augmenting paths of length $4\lceil \log n \rceil + 1$
    ▷ *by Lemma 3, x_i is client-perfect*
6   let A_i be the assignment induced by x_i
7   assign each $c \in C_i$ to $A_i(c)$
8 **end**

a $(1, \lceil 2^{-i}\tau_i \rceil)$-matching. We now compute a $(1, 2\lceil 2^{-i}\tau_i \rceil)$-matching x_i with no augmenting paths of length $4\lceil \log n \rceil + 1$ or smaller. By Lemma 3, each x_i is client-perfect, inducing an (integral) assignment A_i in G_i. Lastly, each client in C_i assigns itself in accordance with A_i to produce the global assignment A. See Algorithm 2.

To formalize the logic of the algorithm, we make a few claims that together will imply the algorithm's correctness. The first claim ensures that the algorithm produces a proper assignment.

▷ Claim 17. Algorithm 2 assigns every client to some server.

Proof. We need to show that the matching x_i computed in Line 5 of Algorithm 2 is client-perfect. Consider τ_i from Line 4 of Algorithm 2. Viewing $\tilde{A}_i$ as a client-perfect $(w, L_{\tilde{A}_i})$-matching, Lemma 16 guarantees that there is an assignment wherein each server s has load at most $\tau_i(s)$.

Because all clients in G_i have the same weight, we can interpret the assignment from Lemma 16 as a client-perfect $(1, \lceil 2^{-i}\tau_i \rceil)$-matching in the unweighted graph G_i. When treating clients as unweighted, server capacities are always bounded by n, and so by Lemma 3, if x_i has no augmenting paths of length $\leq 4\lceil \log n \rceil + 1$, it follows that x_i is client-perfect. ◁

The next claim shows that the assignment produced is $O(1)$-approximate.

▷ Claim 18. There is a universal constant C such that for all $p \geq 1$, including $p = \infty$, the assignment A produced by Algorithm 2 satisfies $\|A\|_p \leq C \|A^*\|_p$, where A^* is an ℓ_p-norm-minimizing assignment.

Proof. Fix $p \geq 1$ (including $p = \infty$). Let A^* and $\tilde{A}^*$ be assignments for G and $\tilde{G}$, respectively, that minimize the ℓ_p-norm. For brevity, we omit the subscript p when writing norms with the understanding that all norms in this proof are ℓ_p-norms. We will also treat the client weight function w as a vector so that we can write its norm as $\|w\|$.

Our strategy is to decompose the final assignment A into two parts and bound the norms of those parts separately. First, we decompose each assignment A_i of Line 6. We define the first part, ρ_i, by $\rho_i(s) = 2^i$ if s is assigned to by A_i and $\rho_i(s) = 0$ otherwise. In other words, ρ_i has the same support as the load vector L_{A_i} of A_i, but all of its nonzero entries are 2^i. The second part, μ_i, docks 2^{i+1} from the support of L_{A_i}: $\mu_i = L_{A_i} - 2\rho_i$. Letting $\mu = \sum_i \mu_i$

and $\rho = \sum_i \rho_i$, we have that $\|A\| = \|\mu + 2\rho\| \le \|\mu\| + 2\|\rho\|$. It therefore suffices to show that $\|\mu\|$ and $\|\rho\|$ both $O(1)$-approximate $\|A^*\|$.

Let us first bound $\|\mu\|$. For any server s assigned to by A_i, we have

$$\begin{aligned} \mu_i(s) &= L_{A_i}(s) - 2^{i+1} \\ &\le 2^{i+1}\lceil 2^{-i}\tau_i(s)\rceil - 2^{i+1} \\ &\le 2^{i+1}\lceil 2^{-i}L_{\tilde{A}_i}(s) + 1\rceil - 2^{i+1} \\ &\le 2^{i+1}2^{-i+1}L_{\tilde{A}_i}(s) + 2^{i+1} - 2^{i+1} && (\lceil x\rceil \le 2x \text{ for all } x \ge 1) \\ &\le 4L_{\tilde{A}_i}(s). \end{aligned}$$

For any server s *not* assigned to by A_i we have $\mu_i(s) = 0$, and so trivially $\mu_i(s) \le 4L_{\tilde{A}_i(s)}$ for such s. Therefore, $\mu(s) = \sum_i \mu_i(s) \le \sum_i 4L_{\tilde{A}_i}(s) = 4L_{\tilde{A}}(s)$. Using Theorem 6, it follows that $\|\mu\| \le 4\|\tilde{A}\| \le 32\|\tilde{A}^*\| \le 32\|A^*\|$.

We now bound $\|\rho\|$. Define $\rho^*(s) = \max_{c \in A^{-1}(s)} w(c)$. Note that ρ^* is the load vector of a "partial" assignment (not all clients are assigned) that assigns to each server at most once. Since w can be interpreted as the load vector of an assignment that assigns *every* client to a unique server, we have $\|\rho^*\| \le \|w\|$. Now observe that $\rho(s) = \sum_{i=1}^{\log \rho^*(s)} \rho_i(s) \le 2\rho^*(s)$ simply because $\rho_i(s)$ is either 0 or 2^i for each i. To complete the bound, notice that $\|w\| \le \|A^*\|$; the best (hypothetical) assignment would assign every client to a unique server, resulting in value $\|w\|$. Putting things together, we have shown that $\|\rho\| \le 2\|A^*\|$. ◁

It remains to bound the round-complexity of the algorithm.

▷ Claim 19. Algorithm 2 takes $O(\log^3 n)$ rounds in the LOCAL model.

Proof. In the LOCAL model, we can easily emulate Algorithm 1 (or any algorithm) on the client-expansion $\tilde{G}$ at no extra cost; any communication across an edge cs simply needs to specify which c_i in the expansion the message is to/from. Since $W \le n$, Algorithm 1 still runs in $O(\log^3 n)$ rounds in the LOCAL model (see Remark 7). The main for-loop is run in parallel, and so we only need to bound the round-complexity of its body. Line 5 is the only line inside the loop that requires (additional) communication, and this again only takes $O(\log^3 n)$ rounds. The total round-complexity is therefore $O(\log^3 n)$. ◁

This concludes the proof of Theorem 15.

5.2 An $O(1)$-approximate $O(m \log^3 n)$-time sequential algorithm

We now show that our approach can also be used to compute an $O(1)$-approximation to the weighted all-norm load balancing problem in near-linear time in the standard sequential setting, proving the following theorem.

▶ **Theorem 20.** *In the standard sequential model, there is a deterministic algorithm to compute an $O(1)$-approximate solution to the weighted all-norm load balancing problem that runs in $O(m \log^3 n)$ time.*

Previously, Azar et al. [2] showed a 2-approximate algorithm for this problem, which runs in two phases: (1) compute an optimal fractional assignment and (2) round the fractional assignment, which incurs a 2-approximation. But their algorithm computes the optimal fractional assignment using the ellipsoid method to solve a linear program with exponentially many constraints, and hence incurs a large polynomial runtime.

Our algorithm uses the same rounding procedure as [2], but instead of computing an exact fractional assignment, we compute an $O(1)$-approximate split assignment in near-linear time by simulating our distributed approach in the sequential setting. To this end, we will need the following subroutine:

▶ **Lemma 21.** *Given any bipartite graph $G = (C \cup S, E)$ and capacity functions κ, τ, it is possible to compute a (κ, τ)-matching with no augmenting paths of length $\leq 8\log(n)$ in $O(m \log^2 n)$ time in the sequential setting.*

Proof. Note that a (κ, τ)-matching corresponds to the following flow problem. Every edge in E gets infinite capacity; there is a dummy source v_s and for every client $c \in C$ there is an edge (v_s, c) of capacity $\kappa(c)$; there is also a dummy sink v_t and for every server $s \in S$ there is an edge from s to v_t of capacity $\tau(s)$. It is immediate to verify that any v_s-v_t flow in this network corresponds to a (κ, τ)-matching and vice versa.

We now show how to compute a solution to this flow problem that contains no augmenting paths of length $9\log(n) \geq 8\log(n) + 2$ which corresponds to the desired (κ, τ)-matching.

The algorithm simply runs $9\log(n)$ successive iterations of blocking flow. A blocking flow in a capacitated graph can be computed in $O(m \log n)$ time using the dynamic tree structure of Sleator and Tarjan [24]. ◀

We are now ready to show our algorithm to compute a split assignment.

▶ **Lemma 22.** *Let $G = (C \cup S, E)$ be a bipartite graph with client-weights $w(C)$. There exists a sequential algorithm that in $O(m \log^3 n)$ time computes a split assignment y_f such that $\left\|L_{y_f}\right\|_p \leq 8 \left\|L_{y_f^*}\right\|_p$ for every $p \geq 1$ (including $p = \infty$).*

Proof. Recall from Observation 14 that the optimal split assignment y_f^* corresponds to an optimal (integral) assignment $\tilde{A}^*$ in the client-expanded graph $\tilde{G}$; server loads in the two solutions are the same, so $\left\|L_{y_f^*}\right\|_p = \|L_{\tilde{A}^*}\|_p$. We obtain our split assignment y_f by simulating Algorithm 1 on the graph $\tilde{G}$: by Theorem 11, this yields the desired 8-approximation. We now describe how to execute the simulation in the sequential model and how to convert between the perspectives of split assignment in G and integral assignment in $\tilde{G}$.

Firstly, in Line 2 of Algorithm 1, we need to a compute a $(1, B)$-matching in $\tilde{G}$ with no short augmenting paths. This is equivalent to a (w, B)-matching in G, which we compute in $O(m \log^2 n)$ time using Lemma 21.

Secondly, in Line 4 of Algorithm 1, each client-copy $\tilde{c}$ in $\tilde{G}$ must find the minimum B such that $\tilde{c}$ is matched in x_B. We need to convert this line to the language of split assignments. In particular, note that in our sequential simulation, x_B is a (w, B)-matching in G rather than a $(1, B)$-matching in $\tilde{G}$. It is easy to see that the following simulates Line 4. For each client c in G, let $s_B(c)$ be the set of servers incident to c in x_B: if an edge cs has multiplicity α in x_B, then s appears α times in $s_B(c)$. To construct the split assignment y_f, first assign c to the server in $s_1(c)$ (if any). Then assign c to an arbitrary $|s_2(c)| - |s_1(c)|$ servers from $s_2(c)$, an arbitrary $|s_4(c)| - |s_2(c)|$ servers from $s_4(c)$, and more generally an arbitrary $|s_B(c)| - |s_{B/2}(c)|$ servers from $s_B(c)$. It is not hard to check that the resulting split assignment is equivalent to some integral assignment in $\tilde{G}$ formed by executing Line 4 of Algorithm 1 in $\tilde{G}$. It is also easy to see that for each x_B the assignments can be performed in $O(m)$ time, for a total of $O(m \log n)$ time.

The running time of the algorithm is thus dominated by the time for computing matchings x_B. Each takes $O(m \log^2 n)$ time to compute (Lemma 21), and there are $O(\log(nW)) = O(\log n)$ values of B, so the total run-time is $O(m \log^3 n)$. ◀

Finally, we round the split assignment to an integral assignment using the rounding procedure of [2], which is described in the proof of Lemma 16. The rounding procedure has two steps: cycle cancelling and computing a matching in a tree. The second can clearly be done in $O(m)$ sequential time. Cycle cancelling can be done deterministically in $O(m \log n)$ time (see, e.g., [15]). The total time for rounding is thus $O(m \log n)$. (Note that in the distributed setting we only relied on the *existence* of such a rounding procedure, because it is unclear how to implement cycle canceling efficiently in the LOCAL model.)

Following the exact same argument as in [2] or in the proof of Lemma 15 of this paper, since our split assignment was an 8-approximation (Theorem 22), the integral assignment formed by rounding yields a 9-approximation. This concludes the proof of Theorem 20.

References

1. Sepehr Assadi, Aaron Bernstein, and Zachary Langley. Improved bounds for distributed load balancing, 2020. arXiv:2008.04148. URL: `https://arxiv.org/abs/2008.04148`.
2. Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J. Woeginger. All-norm approximation algorithms. *J. Algorithms*, 52(2):120–133, 2004. `doi:10.1016/j.jalgor.2004.02.003`.
3. Leonid Barenboim and Gal Oren. Distributed backup placement in one round and its applications to maximum matching and self-stabilization. In *Proc. 3rd Symposium on Simplicity in Algorithms (SOSA 2020)*, pages 99–105, 2020. `doi:10.1137/1.9781611976014.14`.
4. Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^2 n)$ replacements. *J. ACM*, 66(5):Art. 37, 23, 2019. `doi:10.1145/3344999`.
5. Aaron Bernstein, Tsvi Kopelowitz, Seth Pettie, Ely Porat, and Clifford Stein. Simultaneously load balancing for every p-norm, with reassignments. In *Proc. 8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 51, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2017. `doi:10.4230/LIPIcs.ITCS.2017.51`.
6. J. Bruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Comm. ACM*, 17:382–387, 1974. `doi:10.1145/361011.361064`.
7. Deeparnab Chakrabarty and Chaitanya Swamy. Simpler and better algorithms for minimum-norm load balancing. In *Proc. 27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 27, 12. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019. `doi:10.4230/LIPIcs.ESA.2019.27`.
8. Andrzej Czygrinow, Michal Hanćkowiak, Edyta Szymańska, and Wojciech Wawrzyniak. Distributed 2-approximation algorithm for the semi-matching problem. In *Proc. 26th International Symposium on Distributed Computing (DISC 2012)*, volume 7611 of *LNCS*, pages 210–222. Springer, Heidelberg, 2012. `doi:10.1007/978-3-642-33651-5_15`.
9. Jittat Fakcharoenphol, Bundit Laekhanukit, and Danupon Nanongkai. Faster algorithms for semi-matching problems. *ACM Trans. Algorithms*, 10(3):Art. 14, 23, 2014. `doi:10.1145/2601071`.
10. Martin Gairing, Thomas Lücking, Marios Mavronicolas, and Burkhard Monien. The price of anarchy for restricted parallel links. *Parallel Process. Lett.*, 16(1):117–131, 2006. `doi:10.1142/S0129626406002514`.
11. Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: matching, scheduling, and flows. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 468–479. ACM, 2014. `doi:10.1137/1.9781611973402.35`.
12. Magnús M. Halldórsson, Sven Köhler, Boaz Patt-Shamir, and Dror Rawitz. Distributed backup placement in networks. *Distrib. Comput.*, 31(2):83–98, 2018. `doi:10.1007/s00446-017-0299-x`.
13. Nicholas J. A. Harvey, Richard E. Ladner, László Lovász, and Tami Tamir. Semi-matchings for bipartite graphs and load balancing. *J. Algorithms*, 59(1):53–78, 2006. `doi:10.1016/j.jalgor.2005.01.003`.

14 W. A. Horn. Minimizing average flow time with parallel machines. *Oper. Res.*, 21(3), 1973. doi:10.1287/opre.21.3.846.

15 Donggu Kang and James Payor. Flow rounding, 2015. arXiv:1507.08139. URL: https://arxiv.org/abs/1507.08139.

16 Sven Köhler, Volker Turau, and Gerhard Mentges. Self-stabilizing local k-placement of replicas with minimal variance. In *Proc. 14th Stabilization, Safety, and Security of Distributed Systems (SSS 2012)*, pages 16–30, 2012. doi:10.1016/j.tcs.2015.04.019.

17 Christian Konrad and Adi Rosén. Approximating semi-matchings in streaming and in two-party communication. *ACM Trans. Algorithms*, 12(3):Art. 32, 21, 2016. doi:10.1145/2898960.

18 Anshul Kothari, Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Congestion games, load balancing, and price of anarchy. In *Proc. 1st Combinatorial and Algorithmic Aspects of Networking*, volume 3405 of *LNCS*, pages 13–27. Springer, Berlin, 2005. doi:10.1007/11527954_3.

19 Yixun Lin and Wenhua Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European J. Oper. Res.*, 156(1):261–266, 2004. doi:10.1016/S0377-2217(02)00914-1.

20 Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *J. ACM*, 62(5):Art. 38, 17, 2015. doi:10.1145/2786753.

21 Renita Machado and Sirin Tekinay. A survey of game-theoretic approaches in wireless sensor networks. *Comput. Networks*, 52(16):3047–3061, 2008. doi:10.1016/j.gaceta.2008.07.003.

22 Gal Oren, Leonid Barenboim, and Harel Levin. Distributed fault-tolerant backup-placement in overloaded wireless sensor networks. In *Proc. 9th International Conference on Broadband Communications, Networks, and Systems (BROADNETS 2018)*, pages 212–224, 2018. doi:10.1007/978-3-030-05195-2_21.

23 Narayanan Sadagopan, Mitali Singh, and Bhaskar Krishnamachari. Decentralized utility-based sensor network design. *Mobile Networks and Applications*, 11(3):341–350, 2006. doi:10.1007/s11036-006-5187-8.

24 Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

25 Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Uncoordinated load balancing and congestion games in P2P systems. In *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)*, pages 123–130, 2004. doi:10.1007/978-3-540-30183-7_12.

26 Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Selfish load balancing and atomic congestion games. *Algorithmica*, 47(1):79–96, 2007. doi:10.1007/s00453-006-1211-4.