Journal of Machine Learning Research 24 (2023) 1-46 Submitted 6/22; Published 1/23

Robust Load Balancing with Machine Learned Advice

Sara Ahmadian SAHMADIAN@GOOGLE.COM
Google Research
New York, NY 10011, USA

Hossein Esfandiari ESFANDIARI@GOOGLE.COM
Google Research
New York, NY 10011, USA

Vahab Mirrokni MIRROKNI@GOOGLE.COM
Google Research
New York, NY 10011, USA

Binghui Peng BP2601@COLUMBIA.EDU
Columbia University
New York, NY 10027, USA

Editor: Sebastien Bubeck

Abstract

Motivated by the exploding growth of web-based services and the importance of efficiently managing the computational resources of such systems, we introduce and study a theoretical model for load balancing of very large databases such as commercial search engines. Our model is a more realistic version of the well-received balls-into-bins model with an additional constraint that limits the number of servers that carry each piece of the data. This additional constraint is necessary when, on one hand, the data is so large that we can not copy the whole data on each server. On the other hand, the query response time is so limited that we can not ignore the fact that the number of queries for each piece of the data changes over time, and hence we can not simply split the data over different machines.

In this paper, we develop an almost optimal load balancing algorithm that works given an estimate of the load of each piece of the data. Our algorithm is almost perfectly robust to wrong estimates, to the extent that even when all of the loads are adversarially chosen the performance of our algorithm is $1 - 1/e$, which is provably optimal. Along the way, we develop various techniques for analyzing the balls-into-bins process under certain correlations and build a novel connection with the multiplicative weights update scheme.

Keywords: Online algorithm, learning augmented algorithm design, machine learned prediction, load balancing, balls and bins

1. Introduction

Due to the rapid growth of serving demand, web based services face challenging resource allocation problems in their data centers. Driven by the requirement of sub-second latency response, data centers replicate the data across distributed machines to accommodate serving queries in parallel. With a massive number of real time queries to serve on a daily basis, *load balancing* becomes a critical challenge in resource management and lies in the core of distribution system design.

©2023 Sara Ahmadian, Hossein Esfandiari, Vahab Mirrokni, Binghui Peng.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v24/22-0629.html.

The *balls-into-bins* paradigm is the most fundamental model for load balancing in real-time distributed systems (Richa et al., 2001). In the classical balls-into-bins problem, we model the real-time requests as balls and the servers as bins. In each round, a memory-less allocation algorithm places the incoming balls into one of the bins. The goal is to balance the number of balls assigned to the bins, commonly referred as loads. Constrained by the latency requirement, the algorithm is only allowed to look at the loads of a few bins. The most well-established result in the context of balls-into-bins is the *power of two choices*, or more generally, the *power of d choices* algorithm which looks at d bins uniformly at random in each round and assigns the ball to the bin with the minimum load. The maximum load of all bins is then bounded by $\frac{T}{n} + O\left(\frac{\log\log n}{\log d}\right)$ (Berenbrink et al., 2006), where n is the number of bins and T is the number of balls.

The classical formulation of balls-into-bins is extensively studied in the literature (Azar et al., 1994; Vöcking, 2003; Berenbrink et al., 2006; Talwar and Wieder, 2007; Godfrey, 2008; Dahlgaard et al., 2016; Mirrokni et al., 2018; Aamand et al., 2021; Richa et al., 2001), as it captures several applications including hashing, share memory emulations and jobs allocations. However, in modern distributed service systems such as web search engines, there is a significant gap between the theory and practice. In these modern applications, since the size of the whole data is very large, each piece of the data (e.g. one dataset) is only replicated across a few machines, and the replication is *fixed* throughout the time. As a consequence, a query can only be addressed on servers that hold a copy of its corresponding dataset. This challenges the common assumption that one can choose an arbitrary machine to process a request. Hence, two fundamental questions arise:

Q1. How do we replicate the data across servers?

Q2. When a real-time query appears, how should it be assigned?

Given the emerging popularity of web based services, we believe it is crucial to address these questions and mitigate the gap of the classical theory.

In this work, we introduce a new balls-into-bins model that captures the practical issues arising in such distributed systems and develop a near optimal algorithm with machine learned advice.

1.1 Models and motivations

We first provide a high level overview on practical distributed service systems, abstract the model and explain the motivation.

Distributed service system A modern distributed service system can be abstracted into three layers. The first layer consists of end users, the second layer consists of machines called clients, and the third layer consists of machines called servers. We describe the layers from third to first.

The whole data is partitioned into a set of (non-intersecting) datasets. These datasets lie on the servers where each server contains a few of the datasets. Note that, even though the datasets are not intersecting, each dataset may lie on multiple servers. Queries to the datasets are served on the servers. Serving the queries is the time consuming procedure and we intend to balance this load.

For each dataset $\mathcal{D}$ there is one machine called client[1] that is in charge of redirecting the queries to dataset $\mathcal{D}$ to one of the servers that contains a copy of that dataset. The clients are in charge of balancing the loads on the servers. Clients can observe the current loads of the servers to decide which server should serve a query. However, there are only a handful of servers that carry each dataset and a query to a dataset must be served by a server that carries that dataset.

A user may request to perform a query to a dataset $\mathcal{D}$. The query will be sent to the specific client that is associated with the dataset $\mathcal{D}$, to be redirected to a server.

Load balancing as a two stage optimization problem We face two optimization tasks in sequence. The first optimization problem is to decide which dataset to put on which server. This decision should be made before the system goes live. The second optimization problem is to decide how to allocate each query to one of the servers that can serve that query. This optimization problem deals with real time queries, when the system is live. Next, we provide more details on these two stages.

Recall that, the data is partitioned into a set of (non-intersecting) datasets, and we intend to keep a copy of each dataset on only a handful of servers. We use parameter d to refer to the upper-bound on the number of servers that carry each dataset.[2] We call this upper-bound *budget constraint*. We can represent this with a bipartite graph between clients and servers, where an edge between a client A and a server B means that server B contains a copy of the dataset corresponding to client A. Hence, each client can send its queries to its neighboring server in the bipartite graph. The construction of this graph should be done before the system goes live, and is fixed throughout the real-time service. We refer to this as a *network design* problem.

In the second stage, the queries come in a real-time fashion and the clients assign each query to one of the servers that carry the query's corresponding dataset. In fact, our goal in this stage is to balance the load of the servers. The clients can use the information of the previous queries to make the assignment, however, they are not aware of the future queries. We refer to this as an *online load balancing* problem.

Modeling the queries with ML advice There are two common approaches to model the pattern of real-time queries. The first approach, a.k.a. stochastic model (Feldman et al., 2009; Correa et al., 2017; Gupta et al., 2019; Huang and Shu, 2021), assumes that queries are generated by an i.i.d. distribution over the space of all queries. The second approach, all the way on the other end of the spectrum, has no assumption on the arrival of the queries and expects the worst case sequence of queries, a.k.a. adversarial model (Karp et al., 1990; Cohen et al., 2019; Huang et al., 2020). We consider a general model that captures both adversarial and stochastic models. Indeed, we intend to design an algorithm that takes a distribution as a signal, and performs similarly to the best stochastic algorithm when the input matches the signal. Moreover, even if the input is totally adversarial the algorithm is still robust and performs similarly to the best adversarial algorithm. We interpolate the performance of our algorithm based on an unknown parameter $\lambda \in [0, 1]$ that indicates the

1. We call these machines clients because servers are in charge of responding to these machines. In other words, these are the clients to the servers.
2. Note that, unlike our model, in the original balls-into-bins problem each server should be able to process all queries, and hence all of the datasets should be copied on all servers.

similarity of the query sequence to the signal. This model is called algorithm design with ML advice and has drawn a lot of attention in the recent years (Lykouris and Vassilvtiskii, 2018; Rohatgi, 2020; Boyar et al., 2016; Indyk et al., 2019).

1.2 Our Results

We present a near optimal algorithm for the load balancing problem. The performance of our algorithm is measured w.r.t. the *offline optimal solution*, which is defined as the solution that minimizes the maximum load of the servers on the actual load distribution.

Theorem 1 *Suppose $\epsilon > 0$, $d = \Omega(\log n/\epsilon^2 + 1/\epsilon^4)$, and the estimation error is bounded by $\lambda \in [0, 1]$. There is a randomized network construction and online load balancing algorithm such that with probability at least $1 - 1/\operatorname{poly}(n)$,*

- *The maximum load on servers is always $O(1)$ approximate to the offline optimal.*
- *When $T = \Omega\left(\frac{n^2 \log n}{\epsilon^2}\right)$, the maximum load on servers is $(\frac{1}{1-\lambda \exp(-\lambda)} + \epsilon)$ approximate to the offline optimal. Moreover, no algorithm is capable of achieving $(\frac{1}{1-\lambda \exp(-\lambda)} - \epsilon)$ approximation, even when all online requests are revealed at the beginning of the second stage.*

Remark 2 *Based on the total number of balls (requests), there are two natural regimes of the balls-into-bins problem. In the small load regime, the total number of balls (requests) $T = \Theta(n)$, and the hope is to get a constant (or near constant) approximation, e.g. (Richa et al., 2001; Vöcking, 2003). A simple inductive argument extends this constant (or near constant) approximation to all ranges of T. In the large load, or the asymptotic regime, the hope is to get $(1 + o(1))$-approximation (Mitzenmacher and Upfal, 2005), or additive approximation (Berenbrink et al., 2006), when $T \to \infty$. In particular, this would imply the load balancing algorithm converges to an optimal allocation asymptotically.*

Remark 3 *As we will explain soon, the power of d-choice algorithm still guarantees $(1 + o(1))$ approximation asymptotically in the online load balancing stage (Theorem 5). What prevents us from getting $(1 + o(1))$ approximation is the network construction stage, where inaccurate estimation, or adversarial loads, imposes information theoretical barriers (see Theorem 4).*

Our algorithm consists of two subroutines. First, we come up with a network construction algorithm that receives an estimation of future loads as the input, outputs a bipartite graph of clients and servers with near optimal approximation guarantee. Since no online requests are revealed in the network construction stage, the performance of the algorithm is defined as the optimal value one can achieve with the bipartite graph, which is clearly the best one can hope in later online stage, see Section 2 for details.

Theorem 4 (Informal) *For any $\epsilon > 0$, suppose $d \geq \Omega\left(\log n/\epsilon^2 + 1/\epsilon^4\right)$ and the estimation error is bounded by $\lambda \in [0, 1]$. There is a randomized network construction algorithm, such that with probability $1 - 1/\operatorname{poly}(n)$, it achieves $(\frac{1}{1-\lambda \exp(-\lambda)} + \epsilon)$ approximation to the offline optimal. The approximation is tight as no algorithm can achieve $(\frac{1}{1-\lambda \exp(-\lambda)} - \epsilon)$ approximation.*

The second subroutine is online load balancing, where we use the power of d choice (abbreviated as d-choice) algorithm. Fixing the bipartite graph constructed by the algorithm of Theorem 4, we prove the d-choice algorithm is constant approximation in the small load regime and $(1 + o(1))$ approximation asymptotically.

Theorem 5 (Informal) *Let $d = \Omega(\log n)$. Suppose the bipartite graph is constructed by the algorithm of Theorem 4, and we assign online requests using the d-choice algorithm, then with probability at least $1 - 1/\operatorname{poly}(n)$,*

- *The maximum load on servers is always constant approximate to the offline optimal.*
- *When the number of requests $T = \Omega\left(\frac{n^2 \log n}{\epsilon^2}\right)$, the maximum load on servers is $(1+\epsilon)$ approximate to the optimal allocation on the graph.*

1.3 Related works

Load balancing has been in the core of distributed system design, attracting extensive research effort over past decades (Phillips and Westbrook, 1993; Azar et al., 1994; Azar, 1998; Richa et al., 2001; Im et al., 2018; Azar et al., 2018; Ahmadian et al., 2021). Ours is mostly related to the balls-into-bins paradigm (Azar et al., 1994; Richa et al., 2001; Vöcking, 2003), which is the most well-received model for real-time query load balancing.

Balls-into-bins The balls-into-bins paradigm has been extensively studied in the literature (Azar et al., 1994; Richa et al., 2001; Vöcking, 2003; Berenbrink et al., 2006; Talwar and Wieder, 2007; Godfrey, 2008; Dahlgaard et al., 2016; Aamand et al., 2018; Chen, 2019; Lenzen et al., 2019; Mirrokni et al., 2018; Aamand et al., 2021). Azar et al. (1994) initiate this line of work and prove that the d-choice algorithm gives an approximation of $O(\frac{\log \log n}{\log d})$. Vöcking (2003) shows by adopting a non-asymmetric tie breaking rule and non-uniform selection choice, the maximum load can be further reduced to $O(\frac{\log \log n}{d})$. Subsequent work (Berenbrink et al., 2006; Talwar and Wieder, 2007) generalize the result to weighted loads and additive approximations. In contrast with these work, such (smooth) dependence on d is not achievable in our setting. Since the maximum load of clients is $\Omega(\frac{\log n}{\log \log n})$ w.h.p. even for uniform load distribution, one can not do better than $\widetilde{\Omega}(\frac{\log n}{d \log \log n})$ and the only interesting case is $d = \Omega(\log n)$. The work of Godfrey (2008) is most relevant to us, which considers the case that bins are not chosen uniformly at random and proves that as long as some balance conditions are satisfied, one can achieve $O(1)$ approximation when $d = \Omega(\log n)$. The technique of Godfrey (2008) can only be applied to our setting when the load distribution is uniform. For detailed coverage on the literature, we refer the interested readers to the comprehensive survey of Richa et al. (2001).

Machine learned advice There is a formidable trend of incorporating machine learned advice into algorithmic design (Indyk et al., 2019), which tries to close the gap between traditional algorithm design that deals with worst case inputs, and machine learning algorithms that require perfect stochastic assumptions. In this context we design an algorithm given some information (a.k.a. advice) that can be learned from a portion of the input. We want an algorithm that is robust to errors in the advice. There are a growing number of interesting results in this model for different problems including k-server (Böckenhauer et al.,

2017), online set cover (Dobrev et al., 2017), online scheduling (Lattanzi et al., 2020), bloom filters (Mitzenmacher, 2018), caching (Lykouris and Vassilvtiskii, 2018; Rohatgi, 2020), and many others (Purohit et al., 2018; Boyar et al., 2016; Mitzenmacher and Vassilvitskii, 2020; Gollapudi and Panigrahi, 2019; Balcan et al., 2018; Bamas et al., 2020b,a; Wei and Zhang, 2020).

1.4 Technique overview

We now provide a streamlined overview of our techniques.

Network construction We start from the network construction subroutine. The problem is challenging as the estimate load has no accuracy guarantee and the adversary is allowed to make arbitrary perturbations within bounded TV distance. Moreover, even when the actual load distribution is revealed, the computation task is still NP-hard (see Theorem 13). Existing literature on robust algorithm design with ML advice often offers robustness-consistency guarantee, but most of the time they are not optimal, partial due to the seemingly contradictory fact that algorithms need to utilize the estimation to achieve high performance, but at the same time, it is required to ensure the performance does not degrade much when the estimation is incorrect.

We start with some simple yet crucial observations. On one side, if the estimation is completely wrong (i.e. the adversarial case), a natural idea is to randomly assign clients with servers. Through a cute argument, one can show this is in fact $\frac{e}{e-1}$ approximate to the optimal, when $d = \Omega(\log n)$. On the other end of the spectrum, suppose the estimation is accurate, one can come up with a greedy solution that achieves $\frac{d+1}{d}$ approximation.

We combine these ideas and propose the Randomized greedy algorithm, which achieves (near)-optimal approximation guarantees on all ranges of estimation errors. The Randomized greedy algorithm first greedily assign clients to servers. For clients with leftover budget, it randomly distributes them by connecting to randomly chosen servers. The key challenges lie in the analysis of this simple strategy.

First, an important byproduct of the greedy assignment is that the estimate load for any subsets of clients roughly equals the number of connected servers, up to some scaling factors. We refer this as the *volumetric property*. Next, the random assignment induces a bipartite graph with good *expansion properties*. As one can show that the expansion of the graph provides an upper bound on the approximation ratio, combining these two facts already provides worst case performance guarantee.

The volumetric property and expansion property abstract the intuition of why Randomized greedy works well on two extreme cases. Generalizing this to all ranges of (adversarial) perturbation is the most intriguing part of our proof, and it requires some delicate analysis. We outline the intuition behinds the proof. For any subset of clients, if the estimate load is small, then actual load can not be too large, as the TV distance between estimate and actual load is bounded. The expansion property already guarantees reasonable performance in this case. If the estimate load is large, by the volumetric property, we know that the number of connected servers should be larger than normal, and therefore, guarantees reasonable performance. The full proof follows this intuition, but with some extra technical effort, mainly coming from the fact that the volumetric property holds only for light-loaded clients.

Online load balancing The second subroutine is online load balancing and we analyze the performance of the d-choice algorithm. In contrast to most of the existing literature on the *balls-into-bins paradigm*, the neighbor servers (a.k.a. the choice of bins) of each online request (a.k.a. the ball) are neither random nor independent in our context. It is not (uniformly at) random because the greedy allocation of the first subroutine is deterministic and driven by the estimate load, and this is inevitable in general for any algorithm that utilizes the ML advice. It is not independent because the bipartite graph is fixed in advance, and if two requests appear on the same client, they actually share the same set of neighbor servers. We note there are existing literature dealing with correlations between the choice of servers (Godfrey, 2008; Dahlgaard et al., 2016; Aamand et al., 2018), but their methods do not fit into our context.

When the load is small (i.e. $T = n$), the key idea to overcome the above challenges is to separately analyse the initialization and the online process. We perform fine-grained analysis on the initialization, showing it gives a warm start to the online process most of the time. Combine with the classic *layer-induction* techniques for analyzing random process, we are able to prove a constant approximation. Concretely, we consider the *load score* for each server, which is defined as the expected number of requests that the server receives if all clients assign online requests uniformly at randomly. The load score could be large (e.g. $\Omega(\frac{\log n}{\log \log n})$ with non-negligible chance), but the crucial observation is that they sum to n, and hence, they should be small on average. By a simple Markov inequality, one can show at least half of the servers have load score less than 2, and the key challenge is to show concentrations and prove every client has at least $d/2$ assigned servers of low load scores. This is non-trivial as load scores are not independent, not to mention the (deterministic) greedy allocation. We separately analyse the greedy allocation and the random allocation, and the intuition is that (1) The greedy allocations have limited overlaps, and thus contribute little to the load score. (2) The load scores are *negatively associated* and this allows one to use standard concentration inequality. The fine-grained analysis on initialization condition is unique to our context, but we believe the idea could be of independent interests and useful for other balls-into-bins problems. For the online allocation process, given each client is connected to at least $d/2$ low load score servers, we apply the *layer-induction* technique to inductively prove that there are always at least $d/4$ empty connected servers of each client for the first αn requests, where α is a small constant that only depends on the actual load distribution p. By standard argument, this can be generalized to n requests and deduce constant approximation guarantee.

When the load is large, we prove that after $T = \Omega(\frac{n^2 \log n}{\epsilon^2})$ requests, the d-choice algorithm leads to a $(1+\epsilon)$ approximately optimal allocation. Instead of conducting probabilistic analysis on the stochastic process (as usually done in the balls-into-bins literature), we discover an elegant connection between the power of d-choice algorithm and the multiplicative weights updating (MWU) scheme under the i.i.d. arrival model. Based on this connection, we consider a two player zero-sum game between the allocation algorithm and a "load" player, where the allocation algorithm seeks to minimize the load whereas the load player tries to maximize it. One can view the load player uses the MWU strategy while the allocation player always best response. Thus, over time, the value would converge to the minmax value of the game. At the same time, since the requests are drawn i.i.d from the distribution, one can show the minmax value approaches the maximum load under optimal assignment,

via the Azuma-Hoeffding bound. We believe this connection with MWU scheme could be of independent interests.

Organization of the paper

Notations and mathematical formulations are provided in Section 2. Section 3 is devoted to present results on network construction and prove Theorem 4. We provide the analysis of d-choice algorithm in Section 4 and prove Theorem 5. We perform empirical study in Section 5. Proofs are deferred into Appendix.

2. Preliminary

Notation Let $[n] = \{1, \ldots, n\}$, and $[n_1 : n_2] = \{n_1, \ldots, n_2\}$. Let Δ_n be the n dimension simplex that contains all probability distributions over $[n]$. For any distribution $p, q \in \Delta_n$, the total variation (TV) distance between p and q is defined as $\mathrm{TV}(p, q) = \max_{S \subseteq [n]} |p(S) - q(S)|$, where $p(S) = \sum_{i \in S} p_i$ for any subset $S \subseteq [n]$. For any set X, Y, let $X \backslash Y := \{i : i \in X, i \notin Y\}$. We use A to denote clients and B to denote servers. Given a bipartite graph $G(A, B, E)$ between clients and servers, for any subset of clients $S \subseteq A$, we use $N(S)$ to denote the neighbor servers of clients S, i.e., $N(S) := \{j : j \in B, \exists\, i \in S, (i, j) \in E\}$.

2.1 Problem formulation

The load balancing task divides into two stages: The network design stage is devoted to replicate datasets and construct the storage graph between clients and servers. The graph is fixed afterwards, and the online load balancing stage is devoted to assign online requests to servers.

Network design The input consists of (A, B, q, d), where A denotes the set of clients, B denotes the set of servers, $q \in \Delta_{|A|}$ is an estimation of the load distribution over clients, and $d \in \mathbb{N}$ is the budget constraint, meaning that each client can be connected to at most d servers. For simplicity, we assume $|A| = |B| = n$, although all of our results extend naturally to the general case. Let $p \in \Delta_n$ be the actual load distribution over clients A and it is oblivious to the network constructed by the algorithm. The estimation error is measured in total variation distance and let $\lambda := \mathrm{TV}(p, q)$. Both λ and p are unknown to the algorithm. We consider the oblivious adversary model, where the actual load distribution is independent of the randomness of our algorithm.

A graph $G = (A, B, E)$ is feasible if the budget constraint is satisfied. For any feasible graph G between clients and servers, consider the following LP

$$\min_{x, L} L \tag{1}$$
$$\sum_{j \in N(i)} x_{i,j} = p_i \,(\forall i \in A), \quad \sum_{i \in N(j)} x_{i,j} \le L \,(\forall j \in B), \quad x_{i,j} \ge 0 \,(\forall i \in A, j \in B).$$

The optimal allocation of LP (1) is called the *optimal flow*, and its objective value is called the *optimal flow value*, denoted as $\mathrm{OPT}_{G,p}$. We further define the *offline optimal solution* as the minimum possible optimal flow value, denoted as $\mathrm{OPT}_p = \min_{\text{feasible } G} \mathrm{OPT}_{G,p}$.

The offline optimal solution should be thought as the best possible (normalized) loads on servers that one can hope to achieve under load distribution p, as fractional allocation is allowed and loads are calculated on expectations. The approximation ratio of the network construction algorithm is measured in this ideal case, and it is defined as $c_{\text{nd}} = \frac{\mathbb{E}[\text{OPT}_{G,p}]}{\text{OPT}_p}$, where the expectation is taken over the randomness of the algorithm.

Online load balancing After constructing a bipartite graph $G = (A, B, E)$ in the first stage, datasets are replicated across servers and the graph is fixed throughout the online load balancing stage. We consider a stochastic model where requests are drawn i.i.d. from the load distribution p. At each time step t, a request is drawn from p and appears on a client $i_t \in A$, and the algorithm assigns the request to one of the neighbor servers of i_t, *immediately and irrevocably*. We assume the request has unit weight.

We focus on the interesting regime of $T = \Omega(n)$. Let $\text{ALG}_{G,p,t}$ denote the maximum load of the servers at time t ($t \in [T]$), under graph G and load distribution p. We measure the performance w.r.t. the optimal flow value $\text{OPT}_{G,p}$, and denoted as $c_{\text{lb}} = \frac{\mathbb{E}[\text{ALG}_{G,p,T}]}{T \cdot \text{OPT}_{G,p}}$. Here the expectation is taken over the randomness of requests and the randomness of our algorithm.

Remark 6 *Note that instead of the empirical loads, the performance is compared w.r.t. the optimal flow value* $\text{OPT}_{G,p}$*, which is measured on the expected load distribution and fractional allocation is allowed. This is a stronger benchmark and used for the convenience of presentation.*

Overall objective The overall performance of our algorithm is measured w.r.t. the offline optimal, and defined as $c = \frac{\mathbb{E}[\text{ALG}_{G,p,T}]}{T \cdot \text{OPT}_p}$. As noted in Remark 6, we compare with the stronger benchmark that allows fractional allocations over the expected load.

3. Network design

We prove the following result for the network design stage.

Theorem 7 (Formal version of Theorem 4) *Let $\epsilon > 0$, $\lambda \in [0, 1]$.*

(1) Suppose $d \geq \Omega\left(\log n/\epsilon^2 + 1/\epsilon^4\right)$. Then there is a randomized network construction algorithm, such that with probability $1 - 1/\operatorname{poly}(n)$, it achieves $(\frac{1}{1-\lambda\exp(-\lambda)} + \epsilon)$ approximation.

(2) Suppose $\lambda > \epsilon$, $\Omega(\epsilon^{-1}) \leq d \leq O(\frac{\epsilon^2 n}{\log n})$, then no algorithm can achieve $(\frac{1}{1-\lambda\exp(-\lambda)} - \epsilon)$ approximation.

We describe our algorithm in Section 3.1 and the analysis in Section 3.2. The lower bound is proved in Section 3.3, together with a NP-hardness result for the perfect knowledge case (i.e. $\lambda = 0$). The detailed proof are deferred to Appendix B.

3.1 Algorithm

A formal description of Randomized greedy is presented in Algorithm 1. Based on the estimate load q, we split clients into two groups, A_{high} and A_{low}, where A_{high} contains large

loaded clients whose loads are greater than d/n, and A_{low} contains the rest clients of small loads (Line 2). Randomized greedy gives rule for connecting clients to servers. For analysis purpose, we also implicitly maintain assignment of loads. Ideally, we want each server receives no more than $1/n$ unit of loads, and we use b_j to denote the leftover budgets of server j (Line 3).

While this goal is too ambitious for those connected to clients A_{high}, we simply assign d different servers for each of them (Line 5), and split the load evenly. For each client in A_{low}, we start with the first server with non-zero leftover budgets, greedily fill up the server and switch to a new server once the budgets are exhausted (Line 18). We repeat the process, until the client has d connected servers or we have assigned all its load (Line 16). In the former case, we split evenly of the client's leftover loads to all neighbor servers. We call any server assigned to a client so far *greedily assigned*. If a client is assigned to $k < d$ servers, we continue and randomly assign the client to $(d-k)$ more servers (Line 11). We call servers assigned in such way *randomly assigned*.

Notation We denote the i-th client in A_{high} (resp. A_{low}) as $A_{\mathsf{high},i}$ (resp. $A_{\mathsf{low},i}$), and we denote the j-th server as B_j. For any subset clients $S_A \subseteq A$, recall we use $N(S_A)$ to denote the set of neighbor servers for clients in S_A. We use $N_G(S_A)$ (resp. $N_R(S_A)$) to denote the set of greedily (resp. randomly) assigned neighbor servers for clients in S_A.

Algorithm 1 Randomized greedy

1: **Input**: Clients A, servers B, budget d, estimate load distribution $q \in \Delta_n$.
2: $A_{\mathsf{high}} \leftarrow \left\{i \in A : q_i > \frac{d}{n}\right\}$, $A_{\mathsf{low}} \leftarrow A \backslash A_{\mathsf{high}}$
3: $b_j = 1/n$, $\forall j \in B$ ▷ Leftover budget of servers
4: **for** $i = 1, \ldots, |A_{\mathsf{high}}|$ **do**
5: Assign client $A_{\mathsf{high},i}$ to servers $B_{(i-1)d+1} \ldots B_{id}$ ▷ Assign high load clients
6: **end for**
7: $j \leftarrow d|A_{\mathsf{high}}| + 1$.
8: **for** $i = 1, \ldots, |A_{\mathsf{low}}|$ **do**
9: **for** $k = 1, \ldots, d$ **do**
10: **if** $q_{A_{\mathsf{low},i}} = 0$ **then**
11: Assign client $A_{\mathsf{low},i}$ to a server uniformly at random ▷ Random assignment
12: **else**
13: Assign client $A_{\mathsf{low},i}$ to server B_j ▷ Greedy assignment
14: **end if**
15: **if** $b_j > q_{A_{\mathsf{low},i}}$ **then**
16: $b_j \leftarrow b_j - q_{A_{\mathsf{low},i}}$, $q_{A_{\mathsf{low},i}} \leftarrow 0$
17: **else**
18: $q_{A_{\mathsf{low},i}} \leftarrow q_{A_{\mathsf{low},i}} - b_j$, $j \leftarrow j + 1$ ▷ Switch to a new server
19: **end if**
20: **end for**
21: **end for**

3.2 Analysis

The main purpose of this section is to prove the first part of Theorem 7, i.e., Randomized greedy achieves $\left(\frac{1}{1-\lambda\exp(-\lambda)}+\epsilon\right)$ approximation w.h.p.

Randomized greedy possesses two important properties, the *volumetric property* and the *expansion property*. We start with the volumetric property, which states that the estimate load for any subset of clients roughly equals the number of connected servers (up to some scaling factors). This holds even if one ignores those randomly assigned servers. In particular, the volumetric property guarantees that when the estimation is very accurate (i.e. $\lambda = 0$), Randomized greedy achieves $(1+\frac{1}{d})$ approximation.

Lemma 8 (Volumetric property) *When the estimation is accurate ($\lambda = 0$),* randomized greedy *achieves $\left(1+\frac{1}{d}\right)$ approximation. Furthermore, consider the subgraph induced by the greedy assignment,*

(i) For client $i \in A_{\mathsf{high}}$, any server in $N_G(i)$ has load at most $\frac{1}{d}\max_{i\in A} p_i$;

(ii) For client $i \in A_{\mathsf{low}}$, any server in $N_G(i)$ has load at most $\left(1+\frac{1}{d}\right)\frac{1}{n}$.

The graph constructed by randomized greedy exhibits good expansion property.

Lemma 9 (Expansion property) *Suppose $\epsilon > 0$ and $d \geq \Omega(\log n/\epsilon^2 + 1/\epsilon^4)$. With probability $1-1/\operatorname{poly}(n)$ over the randomness of* randomized greedy*, for any subset of clients S_A with $|S_A| \leq n/d$, one has*

$$|N(S_A)| \geq (1-O(\epsilon))\left(|N_G(S_A)| + (n-|N_G(S_A)|)\cdot\left(1-\exp\left(\frac{N_G(S_A)-d\cdot|S_A|}{n}\right)\right)\right), \tag{2}$$

and for any subset of clients S_A with $|S_A| > n/d$, one has

$$|N(S_A)| \geq (1-O(\epsilon))\left(|N_G(S_A)| + (n-|N_G(S_A)|)\cdot\left(1-\exp\left(\frac{N_G(S_A)-n}{n}\right)\right)\right). \tag{3}$$

The proof of Lemma 9 relies on the following two crucial observations. First, the greedy assignment itself spreads out and has good expansion properties. Second, for the random assignment, define $\mathcal{E}_j (j \in [n])$ to be the probability event that the server j is connected with some client in S_A. Though $\{\mathcal{E}_j\}_{j\in[n]}$ are not independent in general, we prove they are *negatively associated* , and therefore, one can give a refined bound on expansion via concentration inequalities.

The following Lemma characterizes the optimal flow value, and more importantly, it relates the expansion of graph to the optimal flow value.

Lemma 10 *Given a graph $G(A,B,E)$ and load distribution p, the optimal flow value satisfies*

$$\mathrm{OPT}_{G,p} = \max_{S_A\subseteq A}\frac{p(S_A)}{|N(S_A)|}.$$

We now provide a proof sketch for the first part of Theorem 7. While we focus on a special case, it gives a taste of the whole proof. The full proof can be found at Appendix B.

Proof [Proof Sketch of Theorem 7, Part 1] Let S_A be the set of clients satisfying

$$\mathrm{OPT}_{G,p} = \frac{p(S_A)}{|N(S_A)|}. \tag{4}$$

The existence of such set is guaranteed by Lemma 10. Let $S_B = N(S_A)$ be the neighbor servers of clients in S_A. Let $k_1 = |S_A|$ be the cardinality of S_A and let $k_2 = |N_G(S_A)|$ be the number of greedily assigned neighbors of S_A. In this proof sketch, we assume

(i) All clients in S_A has estimate load less than d/n, i.e. $q_i \le \frac{d}{n}$ for all $i \in S_A$; and,

(ii) The cardinality of S_A is fairly large, i.e. $|S_A| = k_1 > n/d$.

The volumetric property shown in Lemma 8 guarantees that under the estimate load distribution q, the greedy assignment ensures that all servers in $N_G(S_A)$ has load no more than $(1+\frac{1}{d})\frac{1}{n}$. Hence

$$\frac{q(S_A)}{|N_G(S_A)|} = \frac{q(S_A)}{k_2} \le \frac{d+1}{d}\frac{1}{n}. \tag{5}$$

Moreover, since the TV distance between p and q is at most λ, one has

$$p(S_A) \le \lambda + q(S_A) \le \frac{d+1}{d}\frac{1}{n}k_2 + \lambda. \tag{6}$$

The expansion property (Lemma 9) implies that, with probability $1 - 1/\operatorname{poly}(n)$, one has

$$|N(S_A)| \ge (1 - O(\epsilon))\left(k_2 + (n-k_2)\cdot\left(1 - \exp\left(\frac{k_2 - n}{n}\right)\right)\right). \tag{7}$$

Combining Eq. (4)(5)(6)(7) and the fact that $\mathrm{OPT}_p \ge \frac{1}{n}$, one can bound the approximation ratio

$$\begin{aligned} c_{\mathrm{nd}} = \frac{\mathrm{OPT}_{G,p}}{\mathrm{OPT}_p} &\le \frac{p(S_A)/|N(S_A)|}{1/n}. \\ &\le (1+O(\epsilon))\frac{\frac{d+1}{d}k_2 + n\lambda}{k_2 + (n-k_2)\cdot\left(1 - \exp\left(\frac{k_2-n}{n}\right)\right)}. \end{aligned}$$

Optimizing the above expression, one can upper bound c_{nd} by $\left(\frac{1}{1-\lambda\exp(-\lambda)} + O(\epsilon)\right)$. We conclude the proof sketch. ■

Worst case guarantee A direct corollary of Theorem 7 is that even when the estimate is wrong (i.e. the adversarial case), Randomize greedy still guarantees $\left(\frac{e}{e-1} + \epsilon\right)$ approximation w.h.p. We get this by plugging in $\lambda = 1$.

Corollary 11 (Worst case guarantee) *Suppose the actual load distribution is unknown and the estimate is wrong. For any $\epsilon > 0$, suppose $d \geq \Omega\left(\log n/\epsilon^2 + 1/\epsilon^4\right)$. Then with probability $1 - 1/\operatorname{poly}(n)$,* randomized greedy *still achieves $\left(\frac{e}{e-1} + \epsilon\right)$ approximation.*

Bounded storage Though it is not the main focus of this paper, one can adapt the graph returned by Randomize greedy and make the storage well-balanced on each server. In particular, we can ensure that each server is connected to at most $O(d/\epsilon)$ clients (recall in average a server connects to d clients).

Corollary 12 (Bounded storage) *For any $\epsilon > 0, \lambda \in [0, 1]$ and $d \geq \Omega\left(\log n/\epsilon^2 + 1/\epsilon^4\right)$. There is an algorithm that achieves $\left(\frac{1}{1-\lambda\exp(-\lambda)} + \epsilon\right)$ approximation with probability $1 - 1/\operatorname{poly}(n)$ and guarantees that each server is connected to at most $O(d/\epsilon)$ clients.*

3.3 Lower bound

We finish the second part of Theorem 7 and present an information theoretical lower bound on the approximation ratio, parameterized by the estimation error λ. We give a sketch of the construction and the detailed proof can be found in the Appendix B.

Proof [Proof Sketch of Theorem 7, Part 2] We set the estimate load distribution q to be uniform over the first n/d clients and 0 on other clients. Consider the following hard distribution on the actual load p, we randomly remove λ units of load from the first n/d clients and put them into random $\lambda n/d$ clients in $\{n/d+1, \cdots, n\}$. The offline optimal (i.e. OPT_p) is always $1/n$, while by considering the expected size of $N(S_A)$, where S_A is the set of clients with non-zero loads, one can argue that the best possible approximation ratio is $\left(\frac{1}{1-\lambda\exp(-\lambda)} - \epsilon\right)$. ∎

We also provide a computational lower bound, showing that even one knows the actual load distribution (i.e., $\lambda = 0$), the problem is still NP-hard. It is proved through a careful reduction from the subset-sum problem.

Theorem 13 *Suppose the estimation is accurate ($\lambda = 0$) and the budget $d < n^{1-o(1)}$, then it is NP-hard to find the optimal graph and allocation.*

4. Online load balancing

The second subroutine is to assign online requests to servers, with the goal of balancing loads on servers. We analyse the performance of d-choice algorithm under two regimes: The small load regime $T = n$ and the large load regime $T \to \infty$. We note by a standard argument, the approximation guarantee of $T = n$ carries over to all range of T.

Theorem 14 (Formal version of Theorem 5) *Let $d = \Omega(\log n)$. Suppose the bipartite graph is constructed using* Randomize greedy *and we assign online requests with the d-choice algorithm, then with probability $1 - 1/\operatorname{poly}(n)$,*

- *When the number of requests $T = n$, the maximum load on servers is bounded by $O(1) \cdot \left\{1, \frac{n}{d}\max_{i \in C} p_i\right\}$.*

- *When the number of requests* $T = \Omega\left(\frac{n^2 \log n}{\epsilon^2}\right)$*, the maximum load on servers is* $(1+\epsilon)$ *approximate to the optimal allocation.*

4.1 The small load regime

We prove the first part of Theorem 14 with proof details deferred to Appendix D. Let $T = n$, $D = \left\{1, \frac{n}{d} \max_{i \in C} p_i\right\}$. It is clear that the maximum load is at least D.

Bounds at initialization For each server j, define its *load score* as

$$s_j := \sum_{i=1}^{n} \frac{np_i}{d} \cdot \mathbf{1}\{j \in N(i)\}. \tag{8}$$

Intuitively, the load score s_j equals the expected number of requests that server j receives, if every client assigns the request uniformly at random. Let B_{light} contains servers with low load score, i.e.,

$$B_{\mathsf{light}} := \{j : s_j \leq 2 + D\}.$$

The main technical lemma states that with high probability, each client is connected with at least $\left(\frac{1}{2} - \kappa\right) d$ servers in B_{light} given $d \geq \Omega(\log(d)/\kappa^2)$.

Lemma 15 *Let* $\kappa > 0$ *and* $d \geq \Omega(\log(d)/\kappa^2)$*, then with probability* $1 - 1/\operatorname{poly}(n)$*,* $|N(i) \cap B_{\mathsf{light}}| \geq (\frac{1}{2} - \kappa)d$ *holds for every client* $i \in [n]$*,*

Bounds during the process A server is *empty* if it does not receive any request. We prove that with high probability, the maximum load on servers is 1 after the first $\frac{n}{16(2+D)}$ requests arrived. This is obtained by showing that for each client, it has at least $d/4$ empty neighbor servers after the arrival of the first $\frac{n}{16(2+D)}$ requests. By the allocation rule of d-choice algorithm, it implies the maximum load is 1 after $\frac{n}{16(2+D)}$ requests.

Lemma 16 *Let* $\kappa > 0$ *and* $d \geq \Omega(\log(d)/\kappa^2)$*. With probability* $1 - 1/\operatorname{poly}(n)$*, for each client* $i \in [n]$*, after the arrival of the first* $\left(\frac{1}{16} - \kappa\right) \frac{1}{2+D} n$ *requests, there are at least* $\frac{d}{4}$ *empty servers in* $N(i)$*.*

Proof For $t \in [0 : \left(\frac{1}{16} - \kappa\right) \frac{1}{2+D} n]$, we use $\mathcal{E}_t$ to denote the event that for every client $i \in [n]$, $N(i)$ contains at least $d/4$ empty servers at time t. We prove $\Pr[\mathcal{E}_t] \geq 1 - \frac{t}{n^c}$ for some fixed constant by induction.

The base case is trivial as $\Pr[\mathcal{E}_0] = 1$, i.e., every server is empty at the beginning. Suppose the argument holds up to time $k-1$. For any $i \in [n]$, $t \in [k]$, let the random variable $a(i,t) = 1$ if the t-th request goes to $N(i) \cap B_{\mathsf{light}}$ and every client has at least $d/4$ empty neighbor servers at time $t-1$; $a(i,t) = 0$ otherwise. Denote the randomness of round

$\tau(\tau \le t-1)$ to be w_τ, then we have

$$\begin{aligned}
&\Pr\left[a(i,t)|w_1,\ldots,w_{t-1}\right] \\
\le\ & \sum_{i'\in[n]} \Pr[\text{client } i' \text{ assigns request to } N(i)\cap B_{\mathsf{light}}] \cdot \Pr[\text{client } i' \text{ receives a request}] \\
\le\ & \sum_{i'\in[n]} p_{i'} \cdot \frac{|N(i')\cap N(i)\cap B_{\mathsf{light}}|}{d/4} \\
=\ & \sum_{i'\in[n]} \sum_{j\in N(i)\cap B_{\mathsf{light}}} \frac{4p_{i'}}{d} \cdot \mathbf{1}\{j\in N(i')\} \\
=\ & \sum_{j\in N(i)\cap B_{\mathsf{light}}} \sum_{i'\in[n]} \frac{4p_{i'}}{d} \cdot \mathbf{1}\{j\in N(i')\} \\
=\ & \sum_{j\in N(i)\cap B_{\mathsf{light}}} \frac{4s_j}{n} \\
\le\ & \frac{4d}{n}\cdot(2+D).
\end{aligned}$$

The second step uses the fact that there are at least $d/4$ empty servers in $N(i')$ and the d-choice algorithm assigns requests uniformly at random. The fifth step follows from the definition of s_j (see Eq. (8)), and the last step follows from $s_j \le 2+D$ when $j \in N(i)\cap B_{\mathsf{light}} \subseteq B_{\mathsf{light}}$ and $|N(i)\cap B_{\mathsf{light}}| \le d$.

Hence we know that

$$\begin{aligned}
\mathbb{E}\left[\sum_{t=1}^{k} a(i,t)\right] &\le \frac{4d(2+D)}{n}\cdot k \\
&= \frac{4d(2+D)}{n}\cdot\left(\frac{1}{16}-\kappa\right)\frac{1}{2+D}n \\
&= \left(\frac{1}{4}-4\kappa\right)d.
\end{aligned}$$

Since the random sequence $\{a(i,t)\}_{t\in[k]}$ forms a martingale, using Azuma-inequality, one has

$$\Pr\left[\sum_{t=1}^{k} a(i,t) \ge \left(\frac{1}{4}-\kappa\right)d\right] \le \exp(-4(3\kappa)^2 d/2) \le \frac{1}{n^{c+2}}. \tag{9}$$

We use $d \ge (c+2)\kappa^{-2}\log n$ in the last step.

Consequently, we have that

$$\begin{aligned}
\Pr\left[\overline{\mathcal{E}}_k\right] &\leq \Pr[\overline{\mathcal{E}}_{k-1}] + \Pr[\overline{\mathcal{E}_k} \mid \mathcal{E}_{k-1}] \\
&\leq \Pr[\overline{\mathcal{E}}_{k-1}] + \sum_{i=1}^{n} \Pr\left[\sum_{t=1}^{k} a(i,t) \geq \left(\frac{1}{4} - \kappa\right) d \mid \mathcal{E}_{k-1}\right] \\
&\leq \Pr[\overline{\mathcal{E}}_{t-1}] + \sum_{i=1}^{n} \frac{\Pr\left[\sum_{k=1}^{t} a(i,t) \geq \left(\frac{1}{4} - \kappa\right) d\right]}{\Pr\left[\mathcal{E}_{k-1}\right]} \\
&\leq \frac{t-1}{n^c} + n \cdot \frac{1}{n^{c+2}} \cdot \left(1 - \frac{t}{n^c}\right) \\
&\leq \frac{t}{n^c}.
\end{aligned}$$

The second step follows from an union bound over all clients $i \in [n]$ and the fact that $|N(i) \cap B_{\mathsf{light}}| \geq (\frac{1}{2} - \kappa)d$, the fourth step follows from the induction and Eq. (9). We conclude the proof here. ■

Combining Lemma 15, Lemma 16, one can prove the d-choice algorithm yields constant factor approximation when $T = n$, and thus proves the first part of Theorem 14.

4.2 The large load regime

We next move to the second part of Theorem 14 and prove the asymptotic convergence of the d-choice algorithm. We first observe the equivalence between the d-choice algorithm and a MWU updating scheme (Algorithm 2).

Algorithm 2 MWU for load balancing

1: Initialize $r_j(1) = \frac{1}{n}$ for all server j ($j \in [n]$)
2: **for** $t = 1, \ldots, T$ **do**
3: Set $s(i) = \arg\min_{j \in N(i)} r_j(t)$ for all clients $i \in [n]$.
4: Draw a request $i_t \sim p$ and assign it to server $s(i_t)$
5: Update ▷ $L_j(t)$ denotes the load on server j

$$r_j(t+1) = \frac{\exp(\eta L_j(t))}{\sum_{j \in [n]} \exp(\eta L_j(t))}$$

6: **end for**

Lemma 17 *The Algorithm 2 is equivalent to the d-choice algorithm for any $\eta > 0$.*

We can now prove the second part of Theorem 14. We note the proof does not rely on properties of the underlying bipartite graph.

Proof [Proof of Theorem 14, Part 2.]

We use $\mathcal{A} \subseteq \mathbb{R}^{n\times n}$ to denote the polytope of feasible assignments. That is, for any $A \in \mathcal{A}$, it obeys

$$\sum_{j\in N(i)} A_{j,i} = 1 \quad \forall i \in [n]$$
$$A_{j,i} = 0 \quad \forall i \in [n], j \notin N(i)$$

Consider the following zero-sum game: We fix a load distribution p on the clients. The first player, which we refer as the allocation player, commits $A \in \mathcal{A}$ and the second player, which we we refer as the load player, commits a distribution $r \in \Delta_n$ over servers. The loss for the allocation player is $r^\top Ap$ and the load player receives loss $-r^\top Ap$.

First, a few observations. The polytope $\mathcal{A}$ and Δ_n are convex. Moreover, the loss for both players are linear in their actions. Hence, we can apply the Minmax Theorem.

Lemma 18 (Minmax Theorem Neumann (1928)) *The minmax theorem guarantees that*

$$\min_{A\subseteq\mathcal{A}} \max_{r\in\Delta_n} r^\top Ap = \max_{r\in\Delta_n} \min_{A\in\mathcal{A}} r^\top Ap = \mathrm{OPT}_p$$

Proof The first equality directly follows our previous discussion. That is, the strategy space is convex and the loss is linear. To see the second one. Let A be the optimal allocation for p, then it is clear that $r^\top Ap \le \mathrm{OPT}_p$ holds for any $r \in \Delta_n$. While by Lemma 10, we know that $\mathrm{OPT}_p = \max_{S_A\subseteq[n]} \frac{p(S_A)}{|N(S_A)|}$. If we take r to be uniform distribution over $N(S_A)$, then it is clear that $r^\top Ap \ge \mathrm{OPT}_p$, thus concluding the proof of this Lemma. ■

Back to our proof. We use A_t to denote the allocation at time t. In the online process, at time t, the load player commits $r(t) \in \Delta_n$, receives loss $-A_t e_{i_t}$[3] and updates according to the MWU rule. Here $e_{i_t} \in \mathbb{R}^n$ is the one-hot vector with i_t-th entry equals 1 and others equal 0. Then by the regret guarantee of MWU Arora et al. (2012), one has

$$\begin{aligned} -\sum_{t=1}^{T} r(t)^\top A_t e_{i_t} \le & \min_{r\in\Delta_n} \sum_{t=1}^{T} \langle r, -A_t e_{i_t}\rangle + \frac{\log n}{\eta} + \eta T \\ = & -\max_{r\in\Delta_n} \sum_{t=1}^{T} \langle r, A_t e_{i_t}\rangle + 2\sqrt{T\log n}. \end{aligned} \tag{10}$$

The first inequality follows from the regret bound of MWU, we set $\eta = \sqrt{\frac{\log n}{T}}$ in the second step.

Denote $X_t = r(t)^\top A_t e_{i_t}$, then we know that

$$\mathbb{E}[X_t | e_{i_1}, \cdots, e_{i_{t-1}}] = r(t)^\top A_t p$$

3. We remark that A_t depends on $r(t)$ but i_t does not.

and $|X_t| \leq 1$. $X_1, \ldots, X_t$ forms a martingale, by Azuma-Hoeffding bound, we have

$$\begin{aligned}
&\Pr\left[\left|\sum_{t=1}^{T} r(t)^\top A_t e_{i_t} - \sum_{t=1}^{T} r(t)^\top A_t p\right| \geq \Omega(\sqrt{T\log n})\right] \\
&= \Pr\left[\left|\sum_{t=1}^{T} X_t - \mathbb{E}\left[\sum_{t=1}^{T} X_t\right]\right| \geq \Omega(\sqrt{T\log n})\right] \\
&\leq 1/\operatorname{poly}(n). \qquad (11)
\end{aligned}$$

Finally, since the maximum load on servers equals $\max_{r\in\Delta_n} \sum_{t=1}^{T} \langle r, A_t e_{i_t}\rangle$, with probability $1 - 1/\operatorname{poly}(n)$, one has

$$\begin{aligned}
\max_{r\in\Delta_n} \sum_{t=1}^{T} \langle r, A_t e_{i_t}\rangle &\leq \sum_{t=1}^{T} r(t)^\top A_t e_{i_t} + 2\sqrt{T\log n} \\
&\leq \sum_{t=1}^{T} r(t)^\top A_t p + O(\sqrt{T\log n}) \\
&\leq \min_{A\in\mathcal{A}} \sum_{t=1}^{T} r(t)^\top A p + O(\sqrt{T\log n}) \\
&\leq T \max_{r} \min_{A} r^\top A p + O(\sqrt{T\log n}) \\
&= T\cdot \mathrm{OPT}_p + O(\sqrt{T\log n}).
\end{aligned}$$

The first step follows from Eq. (10), the second step follows from Eq. (11). The third step follows from the fact that we choose $A_t \in \mathcal{A}$ that minimizes $r(t)^\top A p$. The last step follows from Lemma 18. Since $\mathrm{OPT}_p \geq 1/n$, we know that after T rounds, the solution is within $\left(1 + \frac{n\sqrt{\log n}}{\sqrt{T}}\right)$ to the optimal. ■

5. Experiment

In this section, we describe different sets of experiments to show the efficiency of d-choice algorithm as well as the effect of the initial graph construction on the performance of the load balancing algorithm. For our experiments, we use two types of data: (a) real query arrivals generated from Google production web search backend, (b) synthetic query arrivals modeled based on well-established distributions.

In our real world experiments, we first fix the underlying graph based on the algorithm of choice which can be the RANDOMIZED GREEDY algorithm, the random allocation algorithm or the greedy allocation algorithm. Then we fix the stream of the incoming requests and use d-choice algorithm to assign these queries. For determining the query stream, we first determine a probability distribution $p(t)$ over clients for each timestamp t and sample a client to receive a query with respect to $p(t)$. In the case of real data, we use the clients' load data on machines and generate $p(t)$ proportional to this load distribution. For synthetic data, we fix a uniform well-known probability distribution for all timestamps.

After fixing the stream of the requests and the underlying graph, we utilize the d-choice algorithm to assign a client's query to its designated servers. We record the maximum server load for d-choice algorithm over time. Additionally, we periodically calculate the lower-bound on the optimal maximum load by calculating the optimal flow and the theoretical lower bound. Note that the theoretical bound at time t is basically the maximum of the average number of balls per bin, i.e., t/n, and maximum over all clients' loads divided by d. We scale these three values by dividing them by t at time t and then plot them. Our experiments validate the effectiveness of our proposed algorithm.

5.1 Power of d-choice algorithm in real world

In this section, we measure the performance of the d-choice algorithm. We perform two sets of experiments: (i) we fix an underlying graph by RANDOMIZED GREEDY and then compare the performance of the d-choice algorithm to optimal flow value as well as theoretical lower-bound on the machine loads, (ii) we generate three different underlying graphs by RANDOMIZED GREEDY algorithm, the random allocation algorithm, and the greedy allocation algorithm and compare the performance of the d-choice algorithm on the same stream of requests in (i) on these three networks.

We look at three 3 independent query logs from data centers located in Asia(AS), Europe(EU), and North America(NA). For each timestamp, we process the load of different clients on machines to estimate the probability $p(t)$ of a client receiving a query at timestamp t. We then use $p(t)$ to sample queries and decide which client receives a query at timestamp t. We also use the initial distribution, i.e., $p(0)$, to construct the input graph and assign all clients to the same number of servers. The graphs have the same number of machines as clients. We keep the constructed graph fixed over the period of simulation despite the fact that the distribution of loads over clients can change over time.

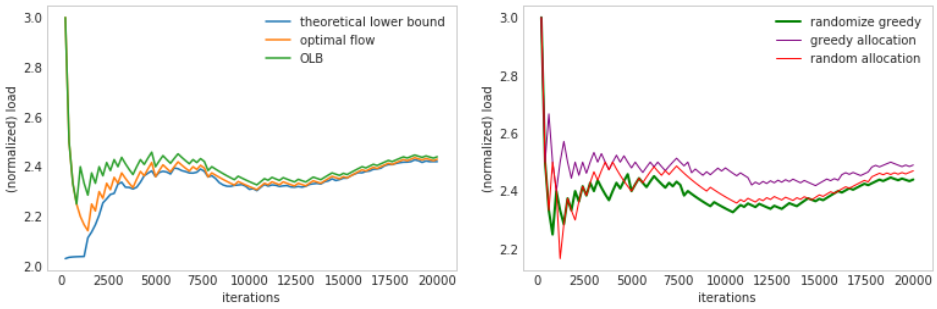
Comparison to theoretical lower-bounds After fixing the underlying graph by RANDOMIZED GREEDY, we apply the d-choice algorithm to queries arrived over time. We process the incoming queries and assign a client's query to one of its designated servers by d-choice algorithm.

During this process, we periodically compute the optimal flow as well as the theoretical lower-bound.[4] The plots on the left hand side of Figure 1 depict the performance of d-choice (denoted as OLB) for three different data sets over time. We can see that as the number of queries grows, for small number of queries, the ratio of max load of servers of the d-choice algorithm to the theoretical lower-bound is at most ≈ 1.5, and it quickly converges to 1 as number of queries grow.

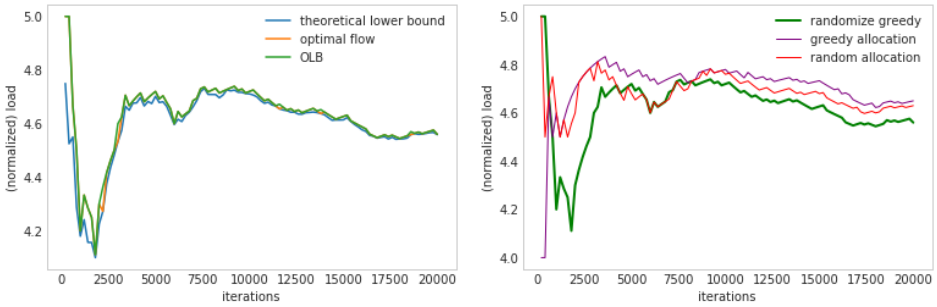
Comparison on different networks While we measure the performance of the d-choice algorithm, we also study the effect of the the underlying graph. This is done by fixing an input graph using RANDOMIZED GREEDY algorithm, the random allocation algorithm, and the greedy allocation algorithm and then running the d-choice algorithm. We use the same query stream in the previous experiments and hence the plot for RANDOMIZED GREEDY is the same as the plot in previous section (Note the y-axis shift). The plots on the right hand

4. We use this theoretical lower bound instead of the theoretical optimal because the later is NP-hard to compute (Theorem 13). Our lower bound is $\left(1 + \frac{1}{d}\right)$ approximate to optimal (Lemma 8).

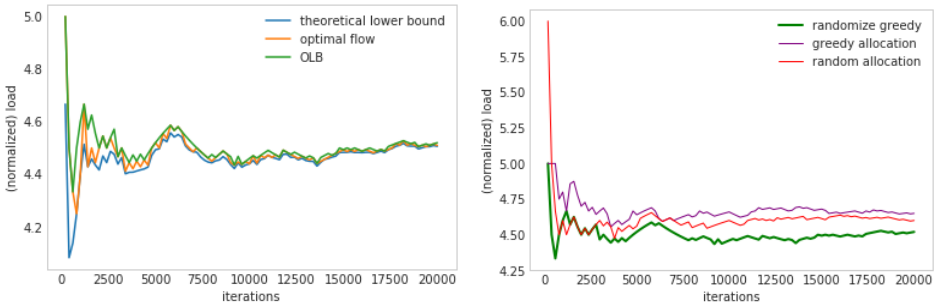
side of 1, show that randomized greedy perform better than both random allocation and greedy allocation for all data sets. However, the difference seems to be within the margin of error.



(a) AS data set



(b) EU data set



(c) NA data set

Figure 1: Performance of d-choice algorithm on real query loads

5.2 Synthetic data

We also perform experiments on the synthetic data. In the experiments, we take 200 clients and 200 servers (i.e. $|A| = |B| = n = 200$) and set the budget constraint $d = 5$.

Comparison theoretical lower-bounds We run experiments on different load distributions and compare the performance of our load balancing algorithm with the optimal flow value of the graph as well as and the trivial theoretical lower bound. We generate

the estimate distribution based on three different families of distributions: the Gaussian distribution, the Exponential distribution and the Multinomial distribution. For the Gaussian (resp. Exponential distribution), we draw each $q_i \sim \mathsf{N}(0, \sigma)$ (resp. $p_i \sim \mathsf{Exp}(\sigma)$) and normalize q, where σ denotes the variance and we set $\sigma = 0.1$ in the experiment; For the Multinomial distribution, we choose a random subset clients S_A of size n/d and set $q_i = d/n$ for each $i \in S_A$. The real load distribution is generated by mixing the estimate distribution together with a perturbation distribution, i.e. $p = (1-\beta)q + \beta\epsilon_{\mathsf{pb}}$, where ϵ_{pb} denotes the perturbation distribution, for which we take a Multinomial distribution. The mixing rate, β, is an upper bound on the total variation distribution between the estimated distribution and the real distribution (i.e. $\lambda \leq \beta$).[5] We vary the mixing rate $\beta = 0, 0.2, 0.5, 1$.

The experiment results can be found in Figure 2 and Table 1. In Figure 2, we plot the load balancing dynamics, where we generate 20000 requests and take a snapshot for each 200 iterations. The experiments validate our theoretical results: (1) the d-choice algorithm has constant approximation when the load is small and it approaches to the optimal flow value asymptotically; (2) the performance of RANDOMIZED GREEDY smoothly changes with the mixing rate. In Table 1, we compute the limiting approximation ratio, this time, we generate 200000 requests and take a median over 9 independent experiments.

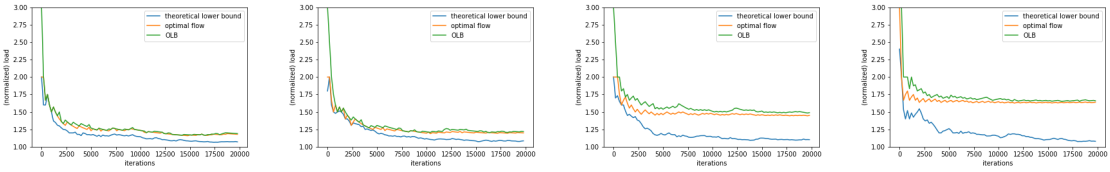
	$\beta = 0$	$\beta = 0.2$	$\beta = 0.5$	$\beta = 1$
Multinomial	1.134	1.275	1.414	1.729
Gaussian	1.01	1.04	1.24	1.62
Exponential	1.03	1.05	1.15	1.66

Table 1: The limiting approximation ratio of d-choice algoirthm

Comparison on different networks We compare the RANDOMIZED GREEDY algorithm with two baseline algorithms: the random allocation algorithm and the greedy allocation algorithm. The random allocation algorithm assigns a random set of servers (of size d) for each client and it is commonly adopted in practice. The greedy allocation algorithm assigns clients to servers using the greedy heuristics (see Line 13 in Algorithm 1). For the leftover budget, it uses a deterministic strategy and makes assignment to the next consecutive servers. In contrast, our RANDOMIZE GREEDY assigns the leftover budget with random allocations (i.e., Line 11 in Algorithm 1). We choose the estimate distribution to be Multinomial distribution and vary the mixing rate in the experiment.

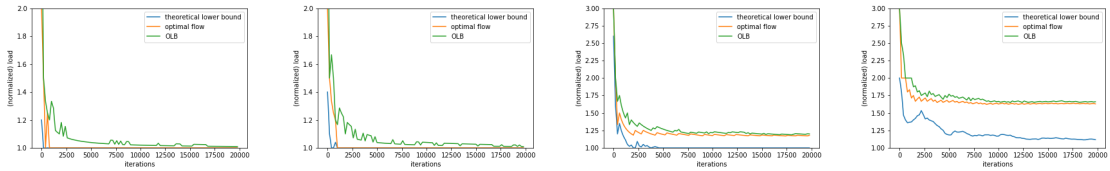
Our experiment results are shown in Figure 4 and Figure 3. In Figure 4, we vary the mixing rate and compare the performance of d-choice algorithm on three set of graphs. We generate 200000 requests for each experiments and we take the median of limiting approximation ratios over 9 independent experiments. For clear presentation, we take logarithmic scale for the y axis in Figure 4. In Figure 3, we plot the load balancing dynamic for three different graphs and take the mixing rate $\beta = 0, 0.2, 0.5, 1$. Again, we generate 20000 requests and take a snapshot for each 200 iterations.

5. We use mixture of distributions (instead of hard total variation perturbation) because it is more natural in practice and easier to generate. Furthermore, for distributions generated from Gaussian, it is impossible to make the TV distance be 1.



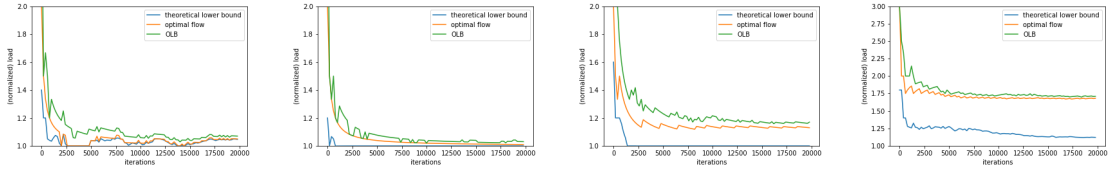
(a) $\beta = 0$ (b) $\beta = 0.2$ (c) $\beta = 0.5$ (d) $\beta = 1$

Multinomial distribution



(e) $\beta = 0$ (f) $\beta = 0.2$ (g) $\beta = 0.5$ (h) $\beta = 1$

Gaussian distribution

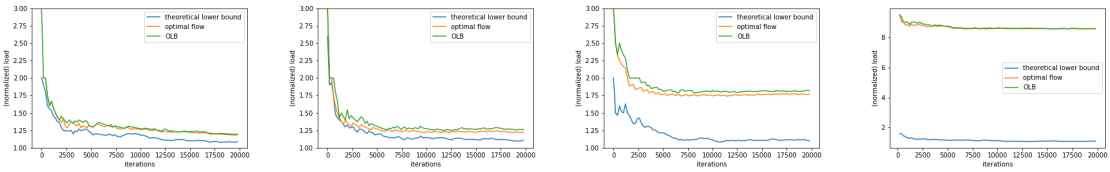


(i) $\beta = 0$ (j) $\beta = 0.2$ (k) $\beta = 0.5$ (l) $\beta = 1$

Exponential distribution

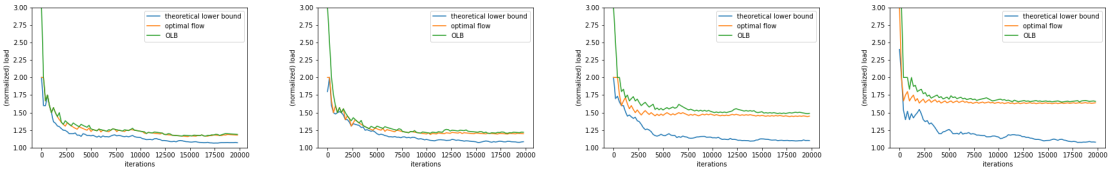
Figure 2: Performance of d-choice on different distributions

These experiments indicate that when $\beta \approx 0$, our algorithm matches the performance of greedy allocation; when $\beta \approx 1$, our algorithm matches the random allocation, while in the middle range, our performance is better than both of them.



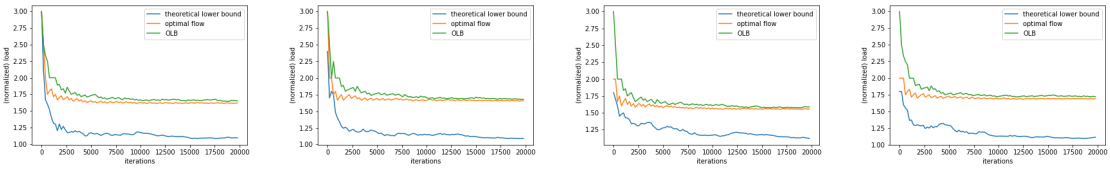
(a) $\beta = 0$ (b) $\beta = 0.2$ (c) $\beta = 0.5$ (d) $\beta = 1$

Greedy allocation



(e) $\beta = 0$ (f) $\beta = 0.2$ (g) $\beta = 0.5$ (h) $\beta = 1$

Randomize Greedy



(i) $\beta = 0$ (j) $\beta = 0.2$ (k) $\beta = 0.5$ (l) $\beta = 1$

Random allocation

Figure 3: Experiments on different graphs

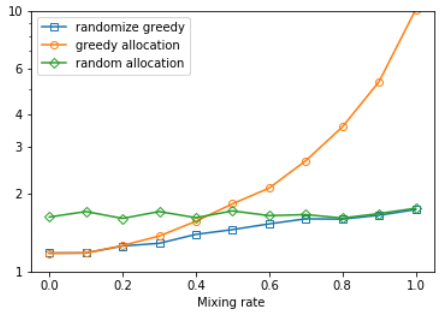
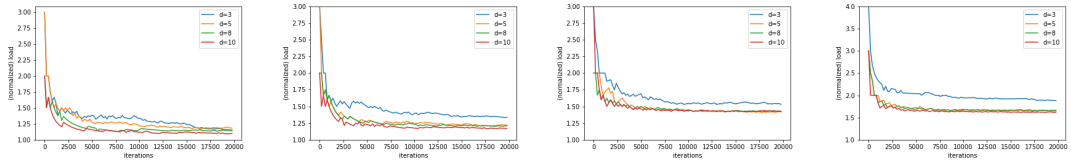


Figure 4: Comparison of three network algorithms

Comparison of different budgets Our theoretical results indicate that as long as the budget d scales logarithmic of the size of the graph, then we can achieve robust and near optimal performance. We further provide experimental supports and exam the effect of budget d on the performance of our algorithm.

We conduct experiments with Multinomial distribution and vary the degree to be $d = 3, 5, 8, 10$. The experiment outcome is shown in figure 5. We observe that as long as we choose $d \geq 5$, the performance is very close, indicating that in practice, the requirement on the budget might be even smaller than the theoretical bound.



(a) $\beta = 0$ (b) $\beta = 0.2$ (c) $\beta = 0.5$ (d) $\beta = 1$

Figure 5: Comparison of different budgets

6. Conclusion

In this paper, we introduce a new model for load balancing by extending the classical balls-into-bins problem that better suits very large databases with massive number of queries such as search engines. Our new model formulates the real-time query load balancing problem as a two-stage optimization task and obtains near optimal algorithm for both stages.

Our paper opens up a number of future directions. First, we assume the online query has unit weight in our paper, a natural open question is to generalize it to arbitrary query weight. Second, we assume the budget $d = \Omega(\log n)$ in our paper, as we explained in Section 1.3, this is necessary to obtain $O(1)$ approximation, but it would be still interesting to obtain $\operatorname{poly}\log(n)$ approximation when the budget $d = O(1)$. Finally, our asymptotic bound of d-choice (i.e., the second part of Theorem 5) makes use of a connection with the MWU scheme. This connection seems to be very general and holds for greedy-type algorithms. It would be interesting to see if it is applicable to other online algorithms.

Acknowledgments

Part of the work was done when Binghui Peng was an intern in Google Research NYC. We wish to thank Cliff Stein for early stage discussion on the project, and suggesting the NP-hardness instance. We thank Hengjie Zhang for useful discussion on the MWU scheme. Binghui Peng is supported in part by Christos Papadimitriou's NSF grants CCF-1763970 AF and CCF-1910700 AF, and Xi Chen's NSF grants CCF-1703925.

References

Anders Aamand, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

Anders Aamand, Jakob Bæk Tejs Knudsen, and Mikkel Thorup. Load balancing with dynamic set of balls and bins. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1262–1275, 2021.

Sara Ahmadian, Allen Liu, Binghui Peng, and Morteza Zadimoghaddam. Distributed load balancing: a new framework and improved guarantees. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

Yossi Azar. On-line load balancing. In *Online algorithms*, pages 178–195. Springer, 1998.

Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. Balanced allocations. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 593–602, 1994.

Yossi Azar, Ilan Reuven Cohen, and Debmalya Panigrahi. Randomized algorithms for online vector load balancing. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 980–991. SIAM, 2018.

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018.

Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *Advances in Neural Information Processing Systems*, 2020a.

Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Advances in Neural Information Processing Systems*, 2020b.

Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM Journal on Computing*, 35(6):1350–1385, 2006.

Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. *Journal of Computer and System Sciences*, 86:159–170, 2017.

Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online algorithms with advice: a survey. *Acm Sigact News*, 47(3):93–129, 2016.

Xue Chen. Derandomized balanced allocation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2513–2526. SIAM, 2019.

Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–25. IEEE, 2019.

José Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. Posted price mechanisms for a random stream of customers. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 169–186, 2017.

Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1631–1642. SIAM, 2016.

Stefan Dobrev, Jeff Edmonds, Dennis Komm, Rastislav Královič, Richard Královič, Sacha Krug, and Tobias Mömke. Improved analysis of the online set cover problem with advice. *Theoretical Computer Science*, 689:96–107, 2017.

Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and Shan Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 117–126. IEEE, 2009.

P Brighten Godfrey. Balls and bins with structure: balanced allocations on hypergraphs. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 511–517, 2008.

Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *International Conference on Machine Learning*, pages 2319–2327, 2019.

Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *International Colloquium on Automata, Languages, and Programming*, 2019.

Zhiyi Huang and Xinkai Shu. Online stochastic matching, poisson arrivals, and the natural linear program. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 682–693, 2021.

Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. Fully online matching. *Journal of the ACM (JACM)*, 67(3):1–25, 2020.

Sungjin Im, Nathaniel Kell, Debmalya Panigrahi, and Maryam Shadloo. Online load balancing on related machines. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 30–43, 2018.

Piotr Indyk, Singer Yaron, Ali Vakilian, and Sergei Vassilvitskii. Summer workshop on learning-based algorithms, 2019. `http://www.mit.edu/~vakilian/ttic-workshop.html`.

Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for online bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.

Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.

Christoph Lenzen, Merav Parter, and Eylon Yogev. Parallel balanced allocations: The heavily loaded case. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 313–322, 2019.

Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.

Vahab Mirrokni, Mikkel Thorup, and Morteza Zadimoghaddam. Consistent hashing with bounded loads. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 587–604. SIAM, 2018.

Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.

Michael Mitzenmacher and Eli Upfal. Probability and computing: Randomized algorithms and probabilistic analysis, 2005.

Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint arXiv:2006.09123*, 2020.

J v Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928.

Steven Phillips and Jeffery Westbrook. Online load balancing and network flow. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 402–411, 1993.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1845. SIAM, 2020.

Kunal Talwar and Udi Wieder. Balanced allocations: the weighted case. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 256–265, 2007.

Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM (JACM)*, 50 (4):568–589, 2003.

Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *Advances in Neural Information Processing Systems*, 2020.

Organization The appendix is organized as follow. We provide probabilistic tools in Appendix A. Missing proof from Section 3 can be found at Appendix B and Appendix C. Missing proof from Section 4 can be found at Appendix D.

Appendix A. Probabilistic tools

Lemma 19 (Chernoff bound) *Let $X = \sum_{i=1}^{n} X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Let $\mu = \mathbb{E}[X] = \sum_{i=1}^{n} p_i$. Then*
1. $\Pr[X \geq (1+\delta)\mu] \leq \exp(-\delta^2\mu/3)$, $\forall \delta > 0$;
2. $\Pr[X \leq (1-\delta)\mu] \leq \exp(-\delta^2\mu/2)$, $\forall 0 < \delta < 1$.

Lemma 20 (Hoeffding bound) *Let $X_1, \cdots, X_n$ denote n independent bounded variables in $[a_i, b_i]$. Let $X = \sum_{i=1}^{n} X_i$, then we have*

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq 2\exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right).$$

Lemma 21 (Azuma-Hoeffding bound) *Let $X_0, \ldots, X_n$ be a martingale sequence with respect to the filter $F_0 \subseteq F_2 \cdots \subseteq F_n$ such that for $Y_i = X_i - X_{i-1}$, $i \in [n]$, we have that $|Y_i| = |X_i - X_{i-1}| \leq c_i$. Then*

$$\Pr[|X_t - Y_0| \geq t] \leq 2\exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right).$$

Lemma 22 (Azuma bound, the multiplicative form) *Let $X_1, \ldots, X_n$ be a set of random variable with $X_i \in \{0, 1\}$ for any $i \in [n]$, and satisfy*

$$\mathbb{E}[X_i | X_1, \ldots, X_n] \leq \mu$$

Then, one has

$$\Pr[\sum_{i=1}^{n} X_i \geq (1+\delta)n\mu] \leq \exp\left(-\delta^2 n\mu/3\right).$$

Definition 23 (Negative association) *A set of random variables $X_1, \ldots, X_n$ is said to be negatively associated (NA) if for any two disjoint sets $I, J \subseteq [n]$ and two functions f, g that are monotone increasing or both monotone decreasing, it holds*

$$\mathbb{E}[f(X_I) \cdot g(X_J)] \leq \mathbb{E}[f(X_I)] \cdot \mathbb{E}[g(X_J)]$$

where we denote $X_I = \{X_i : i \in I\}$

Lemma 24 (Closure property for NA) *We know that*

- *The union of independent sets of NA random variables is NA. That is, if $X_1, \ldots, X_n$ are NA, $Y_1, \ldots, Y_m$ are NA, and $\{X_i\}_{i\in[n]}$ are independent of $\{Y_j\}_{j\in[m]}$, then $X_1, \ldots, X_n$, $Y_1, \ldots, Y_m$ is NA.*

- *Concordant monotone functions defined on disjoint subsets of a set of NA random variables are NA. That is, suppose $f_1, \ldots, f_k : \mathbb{R}^n \to \mathbb{R}$ are all monotonically increasing or all monotone decreasing, where each f_i depends on disjoint subsets of $[n]$, $S_1, S_2, \ldots, S_k \subseteq [n]$. In that case, if $X_1, \ldots, X_n$ are NA, then the set of random variables $Y_1 = f_1(S_1), \ldots, Y_k = f_k(S_k)$ are NA.*

Lemma 25 (Permutation Distributions are NA) *Let $x_1 \leq \cdots \leq x_n$ be n values and let $X_1, \ldots, X_n$ be random variables such that $\{X_1, X_2, \ldots, X_n\} = \{x_1, x_2, \ldots, x_n\}$ always, with all possible assignments equally likely. Then $X_1, X_2, \ldots, X_n$ are NA.*

Lemma 26 (Chernoff-Hoeffding bounds for NA variables) *. Let $X_1, \ldots, X_n$ be NA random variables with $X_i \in [a_i, b_i]$ always. Then $Y = \sum_i X_i$ satisfies Hoeffding's upper tail bound. Namely,*

$$\Pr\left[|Y - \mathbb{E}[Y]| \geq t\right] \leq 2\exp\left(-\frac{2t^2}{\sum_i (a_i - b_i)^2}\right).$$

In particular, if $X_i \in \{0, 1\}$, we have

$$\Pr[Y \geq (1+\delta)\,\mathbb{E}[Y]] \leq \exp\left(-\frac{\delta^2\,\mathbb{E}[Y]}{2+\delta}\right)$$

$$\Pr[Y \leq (1-\delta)\,\mathbb{E}[Y]] \leq \exp\left(-\frac{\delta^2\,\mathbb{E}[Y]}{2}\right).$$

Appendix B. Missing proof from Section 3.2

We first prove Lemma 8, which asserts Randomize greedy achieves $\frac{d+1}{d}$ approximation when the estimate load is accurate ($\lambda = 0$), and more importantly, it gives the volumetric property.

Proof [Proof of Lemma 8] Consider the assignment induced by the Randomize greedy algorithm. Since we assign each client in A_{high} to d different servers and these servers do not receive loads from any other client, we conclude that the maximum load on the them is less than $\frac{1}{d}\max_{i \in A} p_i$. For the rest $n - d|A_{\mathsf{high}}|$ servers, the load can exceed $1/n$ only when a client has d greedily assigned servers and does not distribute all its load. In this case, since the client connects to d servers and the last $(d-1)$ servers it connects with do not connect to any other clients, the leftover load is at most $\frac{d}{n} - (d-1) \cdot \frac{1}{n} = 1/n$. Since we split the leftover load uniformly over its d neighbor servers and these d servers only receive leftover load from this client, the total load on each server is at most $\frac{1}{n} + \frac{1}{n} \cdot \frac{1}{d} = \frac{1}{n} \cdot \frac{d+1}{d}$.

Finally, notice that it is clear that $\mathrm{OPT}_p \geq \frac{1}{n}$, and since the maximum degree is bounded by d, we also have $\mathrm{OPT}_p \geq \frac{1}{d} p_i$ holds for $i \in A$. Thus, the optimal solution satisfies

$$\mathrm{OPT}_p \geq \max\left\{\frac{1}{n}, \frac{1}{d}\max_{i \in A} p_i\right\}.$$

Since no servers in the our assignment has load more than $\max\left\{\frac{1}{n}\frac{d+1}{d}, \frac{1}{d}\max_{i \in A} p_i\right\}$, we conclude it achieves $\left(1 + \frac{1}{d}\right)$ approximation. ■

To prove Lemma 9, we need to following technical result which states the membership of neighborhood is *negative associate*.

Lemma 27 *For any subset of clients $S_A \subseteq A$, Let the random variable X_j ($j \in [n]$) indicate whether the j-th server is connected to some client in S_A, i.e., $X_j = \mathbf{1}_{i \in N(S_A)}$. Then $X_1, \ldots, X_n$ is negative associated.*

Proof For any $i \in S_A, j \in [n]$, define $Y_{i,j} = \mathbf{1}_{j \in N_R(i)}$, that is, $Y_{i,j}$ indicates whether server j is a randomly assigned neighbor of client i. We first show that for any $i \in S_A$, $\{Y_{i,j}\}_j$ is NA. To see this, we further define $Z_{i,j,k} = 1$ if the k-th random edge of client i connects to machine j, where $i \in S_A, j \in [n], 1 \leq k \leq d - n_G(i)$. $\{Z_{i,j,k}\}_j$ is NA since it is a random permutation of $(1, \cdots, 0)$ (see Lemma 25). Furthermore, For any different $k, k' \in [d - n_G(i)]$, $\{Z_{i,j,k}\}_j$ are independent of $\{Z_{i,j,k'}\}_j$, by the closure property of NA (see Lemma 24), we know that $\{Z_{i,j,k}\}_{j,k}$ is NA. Furthermore,

$$Y_{i,j} = \max \left\{ \sum_{k=1}^{d-n_G(i)+1} Z_{i,j,k}, 1 \right\}$$

Since $Y_{i,j}$ ($j \in [n]$) are defined on disjoint sets of $\{Z_{i,j,k}\}_{j,k}$ and the function is monotone, by the closure property of NA (see Lemma 24), we know that $\{Y_{i,j}\}_j$ is NA.

For any different $i, i' \in S_A$, $\{Y_{i,j}\}_j$ are independent of $\{Y_{i',j}\}_j$. By the closure property of NA (see Lemma 24), we know that $\{Y_{i,j}\}_{i,j}$ is NA. Furthermore, we have

$$X_j = \min \left\{ \mathbf{1}_{i \in N_G(S_A)} + \sum_{i \in S_A} Y_{i,j}, 1 \right\}.$$

That is, X_j ($j \in [n]$) are defined on disjoint sets $\{Y_{i,j}\}_{i,j}$ and they are monotone increasing. Hence, by the closure property of NA (see Lemma 24), $X_1, \ldots, X_n$ is NA. ■

We next prove Lemma 9, which is regard the expansion property of Randomize greedy.

Proof [Proof of Lemma 9] Fix a subset of clients $S_A \subseteq A$ and any server $j \in [n]$, we use the random variable X_j to indicate whether server j is a neighbor of client S_A, i.e., $X_j = \mathbf{1}_{i \in N(S_A)}$. For the purpose of proof, we assign a new random server for each client, i.e., we assume there are $(d+1)$ neighbor servers for each client and there is at at least one randomly assigned neighbor servers. This is only used for the proof and we get to remove its effect at the end.

For any server $j \in N_G(S_A)$, we know that $\mathbb{E}[X_j] = 1$. For any server $j \in [n] \backslash N_G(S_A)$, we have

$$\begin{aligned}
\mathbb{E}[X_j] &= 1 - \prod_{i \in S_A} \left(1 - \frac{d+1-N_G(i)}{n}\right) \\
&\geq 1 - \prod_{i \in S_A} \exp\left(-\frac{d+1-N_G(i)}{n}\right) \\
&= 1 - \exp\left(-\frac{\sum_{i \in S_A} d+1-N_G(i)}{n}\right) \\
&\geq 1 - \exp\left(\frac{N_G(S_A) - d \cdot |S_A|}{n}\right) \qquad (12)
\end{aligned}$$

The second step follows from $\exp(-x) \geq 1-x$. For the last step, the overlap between $N_G(i)$ $(i \in S_A)$ counts for at most 1 for each client $i \in S_A$, and therefore, we have

$$N_G(S_A) \geq \sum_{i \in S_A} N_G(i) - |S_A|,$$

and thus,

$$\sum_{i \in S_A} d + 1 - N_G(i) \geq d|S_A| - N_G(S_A).$$

By Lemma 27, we have that $X_1, \cdots, X_n$ is NA and we are going to apply concentration bounds. We divide into cases based on the cardinality of S_A.

Case 1. $|S_A| \leq n/d$.

We have

$$\begin{aligned}
\sum_{j \in [n]} \mathbb{E}[X_j] = & \underbrace{|N_G(S_A)| + (n - |N_G(S_A)|)\left(1 - \exp\left(\frac{N_G(S_A) - d \cdot |S_A|}{n}\right)\right)}_{:=\mathsf{Exps}_1(S_A)} \\
&\geq 0 + n \cdot (1 - \exp(-d|S_A|/n) \\
&\geq \left(1 - \frac{1}{e}\right) d|S_A| \qquad (13)
\end{aligned}$$

The second step holds since the expression is monotone decreasing with $|N_G(S_A)|$ and takes minimum when $|N_G(S_A)| = 0$ (see Lemma 28), the third step follows from $1 - e^{-x} \geq (1 - \frac{1}{e})x$ holds for any $x \in [0, 1]$.

Using Chernoff bound for NA random variable $X_1, \cdots, X_n$ (see Lemma 26), we have

$$\begin{aligned}
\Pr\left[\sum_{j \in [n]} X_j \leq (1-\epsilon) \sum_{j \in [n]} \mathbb{E}[X_j]\right] &\leq \exp\left(-\epsilon^2 \sum_{j \in [n]} \mathbb{E}[X_j]/2\right) \\
&\leq \exp\left(-\epsilon^2 \left(1 - \frac{1}{e}\right) d|S_A|/2\right) \\
&\leq \exp(-\epsilon^2 dk/4), \qquad (14)
\end{aligned}$$

where the second step follows from Eq. (13).

Since we add one more extra neighbor for each clients, this would increase the size of neighbors of S_A by at most $|S_A|$. Thus,

$$N(S_A) \geq \sum_{j\in[n]} X_j - |S_A| \geq \sum_{j\in[n]} X_j - \frac{e}{e-1}\frac{1}{d}\sum_{j\in[n]}\mathbb{E}[X_j] \geq \sum_{j\in[n]} X_j - \frac{2}{d}\sum_{j\in[n]}\mathbb{E}[X_j], \quad (15)$$

where the second inequality follows from Eq. (13).

Consider all subset of clients with size $k \leq n/d$, combining Eq. (12)(14)(15), we have

$$\begin{aligned}
&\Pr\left[\exists S_A \subseteq [n], |S_A| = k, |N(S_A)| \leq \mathsf{Exps}_1(S_A)\right] \\
\leq\ &\Pr\left[\exists S_A \subseteq [n], |S_A| = k, |N(S_A)| \leq \left(1-\epsilon-\frac{2}{d}\right)\sum_{j\in[n]}\mathbb{E}[X_j]\right] \\
\leq\ &\exp(-\epsilon^2 dk/4)\cdot\binom{n}{k} \\
\leq\ &\exp(-\epsilon^2 dk/4 + k\log n) \leq 1/\operatorname{poly}(n)
\end{aligned}$$

The second step follows from the union bound and the last step uses the fact that $d \geq \Omega(\log(n)/\epsilon^2)$.

Case 2. $n/d < |S_A| \leq \frac{1}{2}\epsilon^3 n$.

By Eq. (12), we have

$$\begin{aligned}
\sum_{j\in[n]}\mathbb{E}[X_j] &= |N_G(S_A)| + (n - |N_G(S_A)|)\left(1 - \exp\left(\frac{N_G(S_A) - d\cdot|S_A|}{n}\right)\right) \\
&\geq 0 + n\cdot(1-\exp(-d|S_A|/n) \\
&\geq \left(1-\frac{1}{e}\right)n. \quad (16)
\end{aligned}$$

The second step, again, holds because the expression is monotone increasing with $|N_G(S_A)|$.

Since $X_1, \cdots, X_n$ is NA, by Chernoff bound, we have

$$\begin{aligned}
\Pr\left[\sum_{j\in[n]} X_j \leq (1-\epsilon)\sum_j \mathbb{E}[X_j]\right] &\leq \exp\left(-\epsilon^2\sum_{j\in[n]}\mathbb{E}[X_j]/2\right) \\
&\leq \exp\left(-\epsilon^2\left(1-\frac{1}{e}\right)n/2\right) \\
&\leq \exp(-\epsilon^2 n/4). \quad (17)
\end{aligned}$$

Furthermore, since we add one new random neighbor for each client, we have

$$|N(S_A)| \geq \sum_{i\in S_A} X_i - |S_A| \geq \sum_{i\in S_A} X_i - \frac{1}{2}\epsilon^3 n \geq \sum_{i\in S_A} X_i - \epsilon^3\sum_{i\in S_A}\mathbb{E}[X_i] \quad (18)$$

where the last step holds due to Eq. (16).

Denote

$$\mathsf{Exps}_2(S_A) := \left(|N_G(S_A)| + (n - |N_G(S_A)|) \cdot \left(1 - \exp\left(\frac{N_G(S_A) - n}{n}\right)\right)\right).$$

Consider all subsets of clients of size k ($n/d < k \le \epsilon^3 n/2$), we have

$$\begin{aligned}
& \Pr\left[\exists S_A \subseteq [n], |S_A| = k, |N(S_A)| \le (1-\epsilon-\epsilon^3) \cdot \mathsf{Exps}_2(S_A)\right] \\
\le\ & \Pr\left[\exists S_A \subseteq [n], |S_A| = k, |N(S_A)| \le (1-\epsilon-\epsilon^3) \cdot \mathsf{Exps}_2(S_A)\right] \\
\le\ & \Pr\left[\exists S_A \subseteq [n], |S_A| = k, |N(S_A)| \le (1-\epsilon-\epsilon^3) \sum_{j\in[n]} \mathbb{E}[X_j]\right] \\
\le\ & \exp(-\epsilon^2 n/4) \cdot \binom{n}{k} \\
\le\ & \exp(-\epsilon^2 n/4) \cdot 2^{nH(\epsilon^3/2)} \\
\le\ & \exp(-\epsilon^2 n/8).
\end{aligned}$$

The first step holds since $|S_A| > n/d$, the second step holds due to Eq. (16). The third step follows from the union bound and Eq. (17). The fourth step follows from

$$\binom{n}{k} \le \binom{n}{\epsilon^3 n/2} \le \exp(nH(\epsilon^3/2)),$$

where $H(p) = p\log_2(1/p) + (1-p)\log_2(1/(1-p))$ is the entropy function. The last step follows from

$$H(\epsilon^3/2) = \frac{\epsilon^3}{2} \cdot \log_2(2/\epsilon^3) + \left(1 - \frac{\epsilon^3}{2}\right)\log_2(1/(1-\epsilon^3/2)) \lesssim \epsilon^2/8.$$

Case 3. $|S_A| \ge \epsilon^3 n/2$.

It suffices to prove that with probability at least $1 - 1/\operatorname{poly}(n)$, for any client set S_A with $|S_A| = \epsilon^3 n/2$, $|N(S_A)|$ is greater than $(1-\epsilon)n$. For any set of servers $S_B \subseteq B$ with $|S_B| = \epsilon n$, if none of the servers in S_B is a neighbor of S_A, then all randomly assigned servers from S_A must go to $[n]\backslash S_B$, whose probability is bounded by

$$\Pi_{i\in S_A}(1-\epsilon)^{d-N_G(i)} = (1-\epsilon)^{d|S_A| - \sum_{i\in S_A} N_G(i)} \le (1-\epsilon)^{n/\epsilon} = \exp(-n)$$

The second step follows from $d|S_A| = d\epsilon^3 n/2 \ge n/\epsilon$ and $\sum_{i\in S_A} N_G(i) \le \sum_{i\in[n]} N_G(i) \le 2n$. Taking an union bound over all set S_A and S_B, we conclude the proof. ■

Lemma 10 relates the expansion of the graph to the optimal flow value. We provide the proof here.

Proof [Proof of Lemma 10] First, it is clear that

$$\mathrm{OPT}_{G,p} \ge \frac{p(S_A)}{|N(S_A)|}$$

holds for any set of client $S_A \subseteq [n]$, since the maximum load for servers in $N(S)$ is at least $\frac{p(S_A)}{|N(S_A)|}$ in any feasible allocation.

We delicate to prove the reverse direction. Let S_B denote the set of servers with the maximum load in the optimal allocation, and we further assume S_B has the minimum cardinality among all possible optimal allocations. Let S_A be the set of clients whose neighbors are included in S_B. We claim that servers in S_B do not receive any loads from clients other than S_A. Otherwise, we can shift load from S_B to $B \backslash S_B$ and reduce the cardinality of S_B. This implies that

$$\mathrm{OPT}(G,p) = \frac{p(S_A)}{|N(S_A)|}.$$

Thus concluding the proof. ■

We can now wrap up the proof of the first part (upper bound) of Theorem 7.

Proof [Proof of Theorem 7, Part 1]

Let S_A be the set of clients satisfying

$$\mathrm{OPT}_{G,p} = \frac{p(S_A)}{|N(S_A)|}. \tag{19}$$

The existence of such set is guaranteed by Lemma 10. Let $S_B = N(S_A)$ be the neighbor servers of clients in S_A. Looking forward, one technical subtle of the proof is that we need to divide into cases based on the cardinality of S_A, the intersection of S_A with A_{high}, and the load on S_A. We take moderate effort to simplify the proof but still left with six cases due to subtle difference among them.

We start with a simple case that there is no heavy (estimate) load client in S_A.

No heavy load case. Assume $S_A \cap A_{\mathsf{high}} = \emptyset$.

In another word, we assume clients S_A have estimate load less than d/n, i.e. $q_i \leq \frac{d}{n}$ for all $i \in S_A$. We use k_1 to denote the number of clients in S_A and use k_2 to denote size of greedily assigned servers of clients in S_A. That is, we denote

$$k_1 = |S_A| \quad \text{and} \quad k_2 = |N_G(S_A)|$$

The fact that there is no heavy load client allows us to take advantage of Lemma 8. In particular, the volumetric property shown Lemma 8 guarantees that under the estimate load distribution q, the greedy assignment ensures that all servers in $N_G(S_A)$ have load less than $(1+\frac{1}{d})\frac{1}{n}$. This implies

$$\frac{q(S_A)}{|N_G(S_A)|} = \frac{q(S_A)}{k_2} \leq \frac{d+1}{d}\frac{1}{n}.$$

In fact, we could also assume the equality holds in the above equation. This is w.l.o.g. since it only decreases the gap between $p(S_A)$ and $q(S_A)$, and reduce the distribution shift that is needed. Furthermore, since we only care about the load on clients in S_A, it is also w.l.o.g to assume that the difference between $p(S_A)$ and $q(S_A)$ is exactly λ. Formally, we have

$$q(S_A) = \frac{d+1}{d}\frac{1}{n}k_2 \tag{20}$$

and

$$p(S) - q(S) = \lambda. \tag{21}$$

Our analysis divides into cases based on the cardinality of clients set S_A and the actual load $p(S_A)$.

Case 1. $|S_A| = k_1 > n/d$.

We start with the case that $|S_A|$ is fairly large. Since the total load on S_A can never exceed 1, we know that

$$p(S_A) = \lambda + q(S_A) = \frac{d+1}{d}\frac{1}{n}k_2 + \lambda \leq 1. \tag{22}$$

Using the expansion property shown Lemma 9, one has with probability $1 - 1/\operatorname{poly}(n)$,

$$|N(S_A)| \geq (1 - O(\epsilon)) \left(k_2 + (n - k_2) \cdot \left(1 - \exp\left(\frac{k_2 - n}{n}\right)\right)\right). \tag{23}$$

Since the optimal flow value in any feasible network can not go below $1/n$ (i.e. $\mathrm{OPT}_p \geq 1/n$), the approximation ratio is bounded as

$$c_{\mathrm{nd}} = \frac{\mathrm{OPT}_{G,p}}{\mathrm{OPT}_p} \leq \frac{p(S_A)/|N(S_A)|}{1/n}.$$

Plugging in Eq. (22)(23), we have

$$\begin{aligned}
c_{\mathrm{nd}} &\leq (1 + O(\epsilon)) \frac{\frac{d+1}{d}k_2 + n\lambda}{k_2 + (n - k_2) \cdot \left(1 - \exp\left(\frac{k_2 - n}{n}\right)\right)} \\
&\leq (1 + O(\epsilon)) \frac{\frac{d+1}{d}k_2 + n\lambda}{k_2 + (n - k_2) \cdot \left(1 - \exp\left(\underbrace{-\lambda - \frac{k_2}{dn}}_{-\lambda'}\right)\right)} \\
&= (1 + O(\epsilon)) \frac{k_2 + n\lambda'}{k_2 + (n - k_2) \cdot (1 - \exp(-\lambda'))} \\
&\leq (1 + O(\epsilon)) \frac{n}{n - n\lambda' + n\lambda' \cdot (1 - \exp(-\lambda'))} \\
&= (1 + O(\epsilon)) \frac{1}{1 - \lambda' \exp(-\lambda')} \\
&\leq (1 + O(\epsilon)) \frac{1}{1 - \lambda \exp(-\lambda)}
\end{aligned}$$

where the second step follows from Eq. (22), i.e.

$$\frac{d+1}{d}\frac{1}{n}k_2 + \lambda \leq 1 \quad \Rightarrow \quad k_2 \leq n - n\lambda - \frac{k_2}{d}.$$

In the third step, we take $\lambda' = \lambda + \frac{k_2}{dn}$. The fourth step holds since the expression is monotone increasing of k_2 when fixing λ' (see Lemma 28) and

$$k_2 + n\lambda' = \frac{d+1}{d}k_2 + n\lambda \le n \quad \Rightarrow \quad k_2 \le n - n\lambda'.$$

The last step follows from $\lambda' = \lambda + \frac{k_2}{dn} \le \lambda + 1/d$, $d \ge \Omega(1/\epsilon)$ and Lemma 28.

Case 2. $|S_A| = k_1 \le n/d$.

We next consider the case that $|S_A|$ is small. By the expansion property shown in Lemma 9, with probability at least $1 - 1/\operatorname{poly}(n)$, one has

$$|N(S_A)| \ge (1 - O(\epsilon)) \left(k_2 + (n - k_2) \cdot \left(1 - \exp\left(\frac{k_2 - d \cdot k_1}{n} \right) \right) \right). \tag{24}$$

We further split the discussion based on the magnitude of load $p(S_A)$.

Case 2-a. $|S_A| = k_1 \le n/d$ and $p(S_A) \le \frac{d}{n}|S_A|$.

By Eq. (20)(21), We immediately have

$$p(S_A) = \lambda + q(S_A) = \frac{d+1}{d}\frac{1}{n}k_2 + \lambda \le \frac{d}{n}|S_A| = \frac{d}{n}k_1. \tag{25}$$

Similar to the first case, we have $\mathrm{OPT}_p \ge \frac{1}{n}$, and therefore,

$$c_{\mathrm{nd}} = \frac{\mathrm{OPT}_{G,p}}{\mathrm{OPT}_p} \le \frac{p(S_A)/|N(S_A)|}{1/n}.$$

Plugging in Eq. (24)(25), we have , we have

$$\begin{aligned}
c_{\mathrm{nd}} &\le (1 + O(\epsilon)) \frac{\frac{d+1}{d}k_2 + n\lambda}{k_2 + (n - k_2) \cdot \left(1 - \exp\left(\frac{k_2 - d \cdot k_1}{n} \right) \right)} \\
&\le (1 + O(\epsilon)) \frac{\frac{d+1}{d}k_2 + n\lambda}{k_2 + (n - k_2) \cdot \left(1 - \exp\left(\underbrace{-\lambda - \frac{k_2}{dn}}_{-\lambda'} \right) \right)} \\
&\le (1 + O(\epsilon)) \frac{k_2 + n\lambda'}{k_2 + (n - k_2) \cdot (1 - \exp(-\lambda'))} \\
&\le (1 + O(\epsilon)) \frac{n}{n - n\lambda' + n\lambda'(1 - \exp(-\lambda'))} \\
&= (1 + O(\epsilon)) \frac{1}{1 - \lambda' \exp(-\lambda')} \\
&\le (1 + O(\epsilon)) \frac{1}{1 - \lambda \exp(-\lambda)}.
\end{aligned}$$

The first second step follows from Eq. (24)(25), and the second step follows from Eq. (25). More precisely, we use

$$\frac{d+1}{d}\frac{1}{n}k_2 + \lambda \le \frac{d}{n}k_1 \quad \Rightarrow \quad \frac{k_2 - d \cdot k_1}{n} \le -\lambda - \frac{k_2}{dn}.$$

In the third step, we replace $\lambda' = \lambda + \frac{k_2}{dn}$. The fourth step holds since the expression is monotone increasing of k_2 when fixing λ' (see Lemma 28) and

$$k_2 + n\lambda' = \frac{d+1}{d}k_2 + \lambda = np(S_A) \le n \quad \Rightarrow \quad k_2 \le n - n\lambda'.$$

The last step follows from $\lambda' = \lambda + \frac{k_2}{dn} \le \lambda + 1/d$, $d \ge \Omega(1/\epsilon)$ and Lemma 28.

Case 2-b. $|S_A| = k_1 \le n/d$ and $p(S_A) > \frac{d}{n}|S_A|$.

We next focus on the case that the actual load $p(S_A)$ is fairly large. Since the total load on S_A can not exceed 1, we immediately have

$$p(S_A) = \lambda + q(S_A) = \frac{d+1}{d}\frac{1}{n}k_2 + \lambda \in \left(\frac{d}{n}k_1, 1\right]. \tag{26}$$

Since there are at most $d|S_A|$ neighbors for clients in S_A, we have

$$\mathrm{OPT}_p \ge \frac{p(S_A)}{d|S_A|} = \frac{p(S_A)}{dk_1},$$

and therefore,

$$c_{\mathrm{nd}} \le \frac{\mathrm{OPT}_{G,p}}{\mathrm{OPT}_p} \le \frac{p(S_A)/|N(S_A)|}{p(S_A)/d|S_A|}.$$

Plugging in Eq. (24), we have

$$\begin{aligned}
c_{\mathrm{nd}} &\le (1+O(\epsilon))\frac{dk_1}{k_2 + (n-k_2)\cdot\left(1-\exp\left(\frac{k_2 - d\cdot k_1}{n}\right)\right)} \\
&\le (1+O(\epsilon))\frac{dk_1}{dk_1 - n\lambda - \frac{k_2}{d} + (n - dk_1 + n\lambda + \frac{k_2}{d})\cdot\left(1-\exp\left(\underbrace{-\lambda - \frac{k_2}{dn}}_{-\lambda'}\right)\right)} \\
&= (1+O(\epsilon))\frac{(dk_1 - n\lambda') + n\lambda'}{(dk_1 - n\lambda') + (n - (dk_1 - n\lambda'))\cdot(1-\exp(-\lambda'))} \\
&\le (1+O(\epsilon))\frac{(n - n\lambda') + n\lambda'}{(n - n\lambda') + (n - (n - n\lambda'))\cdot(1-\exp(-\lambda'))} \\
&= (1+O(\epsilon))\frac{1}{1-\lambda'\exp(-\lambda')} \\
&\le (1+O(\epsilon))\frac{1}{1-\lambda\exp(-\lambda)}.
\end{aligned}$$

The second step holds since the denominator is monotone increasing with k_2 and by Eq. (26),

$$\frac{d+1}{d}\frac{1}{n}k_2 + \lambda \ge \frac{d}{n}k_1 \quad \Rightarrow \quad k_2 \ge dk_1 - n\lambda - \frac{k_2}{d}.$$

In the third step, we replace $\lambda' = \lambda + \frac{k_2}{dn}$. The fourth step holds since the expression is monotone increasing of $(dk_1 - n\lambda')$ when fixing λ' (see Lemma 28) and $dk_1 \le n$. The last step follows from follows from $\lambda' = \lambda + \frac{k_2}{dn} \le \lambda + 1/d$, $d \ge \Omega(1/\epsilon)$ and Lemma 28.

We next consider the general case that there are heavy load clients in S_A.

Heavy load case Assume $S_A \cap A_{\mathsf{high}} \neq \emptyset$.

Let $S_{A,\mathsf{high}} = S_A \cap A_{\mathsf{high}}$ and $S_{A,\mathsf{low}} = S_A \cap A_{\mathsf{low}}$. The major difference is that for servers in $N_G(S_{A,\mathsf{high}})$, their load could be much larger than $\left(1 + \frac{1}{d}\right)\frac{1}{n}$. Thus we can not directly apply the volumetric property proved in Lemma 8 and say the estimate load on S_A roughly equals the number of neighbor servers. However, we will see that this won't degrade the performance of our algorithm. The intuition is that all clients in $S_{A,\mathsf{high}}$ are connected to d different servers, which is already the best we can hope.

For any client $i \in S_{A,\mathsf{high}}$, we first argue that it is w.l.o.g. to assume $p(i) = q(i) \le d\,\mathrm{OPT}_{G,p}$. The reason is as follow. (1) If $p_i > q_i$, then we can increase q_i to p_i. (2) If $p_i \le q_i$ and $p_i > d/n$, we can decrease q_i to p_i. (3) If $p_i \le \frac{d}{n} < q_i$, we can decrease $q(i)$ to d/n and put client i in $S_{A,\mathsf{low}}$. In all these three cases, we do not change the execution of our algorithm and the output bipartite remains the same, while the TV distance between p and q decrease. Finally, we take a step further and assume that $p_i = dL > \frac{d}{n}$ holds for all client $i \in S_{A,\mathsf{high}}$. This is w.l.o.g. since we only take account into the total loads on $S_{A,\mathsf{high}}$.

We make slight change of notations and denote

$$|S_{A,\mathsf{low}}| = k_1, \quad |N_G(S_{A,\mathsf{low}})| = k_2, \quad \text{and} \quad |S_{A,\mathsf{high}}| = k_3.$$

We note that

$$|N_G(S_A)| = |N_G(S_{A,\mathsf{high}})| + |N_G(S_{A,\mathsf{low}})| = k_1 + dk_3, \tag{27}$$

since there is no overlap between $N_G(S_{A,\mathsf{low}})$ and $N_G(S_{A,\mathsf{high}})$. Furthermore, we can still assume that

$$p(S_{A,\mathsf{low}}) = q(S_{A,\mathsf{low}}) + \lambda = \frac{d+1}{d}\frac{1}{n}k_2 + \lambda. \tag{28}$$

Our analysis divides into cases based on the size of $|S_A|$ and the actual load $p(S_A)$.

Case 3. $|S_A| \ge n/d$.

By Lemma 9, with probability $1 - 1/\operatorname{poly}(n)$, one has

$$|N(S_A)| \ge dk_3 + k_2 + (n - dk_3 - k_2)\left(1 - \exp\left(\frac{dk_3 + k_2 - n}{n}\right)\right). \tag{29}$$

Furthermore, one has $\mathrm{OPT}_p \ge L$ and

$$p(S_A) = p(S_{A,\mathsf{high}}) + p(S_{A,\mathsf{high}}) = Ldk_3 + \frac{d+1}{d}\frac{1}{n}k_2 + \lambda \le 1 \tag{30}$$

Hence, we have

$$c_{\mathrm{nd}} = \frac{\mathrm{OPT}_{G,p}}{\mathrm{OPT}_p} \le \frac{p(S_A)/|N(S_A)|}{L}.$$

Plugging in Eq. (29)(30), we have

$$\begin{aligned}
c_{\mathrm{nd}} &\le (1-O(\epsilon))\frac{dk_3+\lambda/L+(d+1)k_2/dLn}{dk_3+k_2+(n-dk_3-k_2)\left(1-\exp\left(\frac{dk_3+k_2-n}{n}\right)\right)} \\
&= (1-O(\epsilon))\frac{Ln\cdot(dk_3+\lambda/L+(d+1)k_2/dLn)}{Ln\cdot\left(dk_3+k_2+(n-dk_3-k_2)\left(1-\exp\left(\frac{dk_3+k_2-n}{n}\right)\right)\right)} \\
&< (1-O(\epsilon))\frac{dk_3Ln+n\lambda+(d+1)k_2/d}{dk_3nL+k_2+(n-dk_3nL-k_2)\left(1-\exp\left(dk_3L+\frac{k_2}{n}-1\right)\right)} \\
&\le (1-O(\epsilon))\frac{dk_3Ln+n\lambda+(d+1)k_2/d}{dk_3Ln+k_2+(n-dk_3Ln-k_2)\left(1-\exp\left(\underbrace{-\lambda-\frac{k_2}{dn}}_{-\lambda'}\right)\right)} \\
&= (1-O(\epsilon))\frac{dk_3Ln+k_2+n\lambda'}{dk_3Ln+k_2+(n-dk_3Ln-k_2)\left(1-\exp\left(-\lambda'\right)\right)} \\
&\le (1-O(\epsilon))\frac{1}{1-\lambda'+\lambda'\left(1-\exp\left(-\lambda'\right)\right)} \\
&\lesssim (1-O(\epsilon))\frac{1}{1-\lambda\exp(-\lambda)}.
\end{aligned}$$

The third step follows from the monotonicity of k_2 in the denominator and $nL>1$. The fourth step follows from Eq. (30), that is

$$dk_3L+\frac{d+1}{d}\frac{1}{n}k_2+\lambda\le 1 \quad\Rightarrow\quad dk_3L+\frac{k_2}{n}-1\le -\lambda-\frac{k_2}{dn}.$$

We replace $\lambda'=\lambda+\frac{k_2}{dn}$ in the fifth step and the sixth step holds since the expression is monotone increasing in dk_3nL+k_2 when fixing λ' (see Lemma 28) and by Eq. (30), one has

$$Ldk_3+\frac{d+1}{d}\frac{1}{n}k_2+\lambda\le 1 \quad\Rightarrow\quad dk_3Ln+k_2\le n-n\lambda'.$$

The last step follows from $\lambda'=\lambda+\frac{k_2}{dn}\le\lambda+1/d$, $d\ge\Omega(1/\epsilon)$ and Lemma 28

Case 4. $|S_A|\le n/d$.

By Lemma 9, with probability $1-1/\operatorname{poly}(n)$, we have

$$|N(S_A)|\ge dk_3+k_2+(n-dk_3-k_2)\left(1-\exp\left(\frac{k_2-dk_1}{n}\right)\right). \tag{31}$$

We further split into two cases.

Case 4-a. Suppose the load on $p(S_A)$ is fairly small, i.e.

$$p(S_{A,\mathsf{low}})=\frac{d+1}{d}\frac{1}{n}k_2+\lambda\le\frac{k_1d}{n}. \tag{32}$$

Then the total load on clients S_A obeys

$$p(S_A) = p(S_{A,\mathsf{high}}) + p(S_{A,\mathsf{low}}) = dk_3 L + \frac{d+1}{d}\frac{1}{n}k_2 + \lambda \le 1. \tag{33}$$

Furthermore, since we assume $p_i = dL$ ($L > 1/n$) for all client $i \in S_{A,\mathsf{high}}$ and $S_{A,\mathsf{high}} \neq \emptyset$, we have $\mathrm{OPT}_p \ge L$. Hence,

$$c_{\mathrm{nd}} = \frac{\mathrm{OPT}_{G,p}}{\mathrm{OPT}_p} \le \frac{p(S_A)/|N(S_A)|}{L}.$$

Plugging in Eq. (31)(33), we have

$$\begin{aligned}
c_{\mathrm{nd}} &\le (1-O(\epsilon))\frac{dk_3 + \lambda/L + (d+1)k_2/Lnd}{dk_3 + k_2 + (n - dk_3 - k_2)\left(1 - \exp\left(\frac{k_2 - dk_1}{n}\right)\right)} \\
&\le (1-O(\epsilon))\frac{dk_3 + \lambda/L + (d+1)k_2/Lnd}{dk_3 + k_2 + (n - dk_3 - k_2)\left(1 - \exp\left(\underbrace{-\lambda - \frac{k_2}{dn}}_{-\lambda'}\right)\right)} \\
&= (1-O(\epsilon))\frac{dk_3 + k_2/Ln + \lambda'/L}{dk_3 + k_2/Ln + (n - dk_3 - k_2/nL)\left(1 - \exp\left(-\lambda'\right)\right)} \\
&= (1-O(\epsilon))\frac{Ln\cdot(dk_3 + k_2/Ln + \lambda'/L)}{Ln\cdot(dk_3 + k_2/Ln + (n - dk_3 - k_2/Ln)\left(1 - \exp\left(-\lambda'\right)\right))} \\
&\le (1-O(\epsilon))\frac{Lndk_3 + k_2 + n\lambda'}{Lndk_3 + k_2 + (n - Lndk_3 - k_2)\left(1 - \exp\left(-\lambda'\right)\right)} \\
&\le (1-O(\epsilon))\frac{n - n\lambda'}{n - n\lambda' + n\lambda'\left(1 - \exp\left(-\lambda'\right)\right)} \\
&= (1-O(\epsilon))\frac{1}{1 - \lambda'\exp(-\lambda')} \\
&\lesssim (1-O(\epsilon))\frac{1}{1 - \lambda\exp(-\lambda)}.
\end{aligned}$$

The second step follows from Eq. (32), i.e.

$$p(S_C^Y) = \frac{d+1}{d}\frac{1}{n}k_2 + \lambda \le \frac{k_1 d}{n}. \quad \Rightarrow \quad \frac{k_2 - dk_1}{n} \le -\lambda - \frac{k_2}{dn}.$$

We replace $\lambda' = \lambda + \frac{k_2}{dn}$ in the third step. The sixth step holds since the expression is monotone increasing of $(Lndk_3 + k_2)$ when fixing λ' (see Lemma 28) and

$$Lndk_3 + k_2 + n\lambda' = n\cdot\left(Ldk_3 + \frac{d+1}{d}\frac{1}{n}k_2 + \lambda\right) = np(S_A) \le n \quad \Rightarrow \quad Lndk_3 + k_2 \le n - n\lambda'.$$

The last step follows from $\lambda' = \lambda + \frac{k_2}{dn} \le \lambda + 1/d$, $d \ge \Omega(1/\epsilon)$ and Lemma 28.

Case 4-b. Suppose the load $p(S_{A,\mathsf{low}})$ is fairly large and satisfies

$$p(S_A^Y) = \frac{d+1}{d}\frac{1}{n}k_2 + \lambda > \frac{k_1 d}{n}. \tag{34}$$

Then we have

$$\mathrm{OPT}_p \geq \frac{p(S_A)}{dk_1 + dk_3},$$

and

$$c_{\mathrm{nd}} = \frac{\mathrm{OPT}_{G,p}}{\mathrm{OPT}_p} \leq \frac{p(S_A)/|N(S_A)|}{p(S_A)/(dk_1+dk_3)} = \frac{dk_1+dk_3}{|N(S_A)|}.$$

Plugging in Eq. (31), one has

$$\begin{aligned}
c_{\mathrm{nd}} &\leq (1-O(\epsilon))\frac{dk_1+dk_3}{dk_3+k_2+(n-dk_3-k_2)\left(1-\exp\left(\frac{k_2-dk_1}{n}\right)\right)} \\
&\leq (1-O(\epsilon))\frac{dk_1+dk_3}{dk_3+dk_1-n\lambda-\frac{k_2}{d}+(n-dk_3-dk_1+n\lambda+\frac{k_2}{d})\left(1-\exp\left(\underbrace{-\lambda-\frac{k_2}{dn}}_{-\lambda'}\right)\right)} \\
&\leq (1-O(\epsilon))\frac{dk_1+dk_3-n\lambda'+n\lambda'}{dk_3+dk_1-n\lambda'-+(n-dk_3-dk_1+n\lambda')\left(1-\exp\left(-\lambda'\right)\right)} \\
&\leq (1-O(\epsilon))\frac{1}{1-\lambda'\exp(-\lambda')} \\
&\leq \frac{1}{1-\lambda\exp(-\lambda)}.
\end{aligned}$$

The second step holds since the denominator is monotone increasing with dk_3+k_2 (see Lemma 28) and Eq. (34), that is

$$\frac{d+1}{d}\frac{1}{n}k_2+\lambda > \frac{k_1 d}{n} \quad \Rightarrow \quad dk_3+k_2 \geq dk_3+dk_1-\frac{k_2}{d}-n\lambda.$$

We take $\lambda' = -\lambda + \frac{k_2}{dn}$ in the third step. The fourth step holds since the expression is monotone increasing in $(dk_1+dk_3-n\lambda')$, and by Eq. (34), one has

$$1 \geq p(S_{A,\mathsf{low}})+p(S_{A,\mathsf{high}}) = \frac{d+1}{d}\frac{1}{n}k_2+\lambda+dk_3 L \geq \frac{dk_1}{n}+\frac{dk_3}{n} \quad \Rightarrow \quad dk_1+dk_3-n\lambda' \leq n-n\lambda'.$$

We conclude the proof here. ■

Lemma 28 (Technical Lemma) *We have*

1. *Let $f_1(x) = x+(n-x)(1-\exp(\frac{x}{n}-\alpha))$, where $x \in [0,n]$ and $\alpha \leq 1$. Then $f_1(x)$ is monotone increasing.*

2. *Let* $f_2(x) = \frac{x+n\lambda}{x+(n-x)(1-\exp(\lambda))}$. *Then* $f_2(x)$ *is monotone increasing.*

3. *Let* $f_3(x) = \frac{1}{1-x\exp(-x)}$. *For* $x \in (0,1)$, $\epsilon \in (0, \frac{1}{2})$ *we have* $f_3(x+\epsilon) \leq (1+O(\epsilon))f_3(x)$.

Proof For the first one, we have

$$f_1'(x) = \frac{x}{n}\exp(x/n - \alpha) \geq 0.$$

For the second one, we have

$$f_2'(x) = \frac{n(1-(1+\lambda)\exp(-\lambda)))}{(x+(n-x)(1-\exp(-\lambda)))^2} \geq 0.$$

For the third one, we have

$$f_3'(x) = \frac{\exp(-x)(1-x)}{(1-x\exp(-x))^2}.$$

It is easy to see that $f_3(x)$ is increasing in $(0,1)$ and decreasing in $(1,2)$. Furthermore, $|f_3'(x)| \leq e^2/(e-1)^2 \leq 4$, thus we know

$$f_3(x+\epsilon) - f_3(x) \leq 4\epsilon \approx O(\epsilon)f_3(x).$$

■

Finally, we prove one can modify Randomize greedy and ensure each server is connected to at most $O(d/\epsilon)$ clients, i.e. Corollary 12.

Proof [Proof of Corollary 12] Given a set of client A and a set of server B with $|A| = |B| = n$, we divide the server into two parts $B = B_1 \cup B_2$ with $|B_1| = (1-\epsilon')n$ and $|B_2| = \epsilon' n$. We run Randomize greedy on client set A and server set B_1. By Theorem 7, the Randomize greedy algorithm returns a network that is $\frac{1}{1-\lambda\exp(-\lambda)} + O(\epsilon')$ approximately optimal. We use the set B_2 to balance the storage on servers. In particular, for any server in B_1 that has more than d/ϵ' connected clients, we redirect these clients to servers in B_1. Since the total number of edges between A and B_1 is at most nd and there are $\epsilon' n$ servers in B_2, we can make sure that no server has at more than d/ϵ' neighbors. This new network can only be better than the original one. We conclude the proof by choosing a suitable parameter $\epsilon' = O(\epsilon)$. ■

Appendix C. Missing proof from Section 3.3

Proof [Proof of Theorem 7, Part 2] For simplicity, we assume n is divisible by d. Let the estimate load distribution q be uniform over the first n/d clients, i.e.

$$q = \left(\underbrace{\tfrac{d}{n}, \ldots, \tfrac{d}{n}}_{n/d}, 0, \ldots, 0 \right).$$

Consider the following hard distribution on the actual load. For each $i \in [n/d]$, let $\widetilde{p}_i = \frac{d}{n}$ with probability $(1-\lambda+\epsilon)$ and $\widetilde{p}_i = 0$ otherwise. For each $i \in [n/d+1:n]$, let $\widetilde{p}_i = \frac{d}{n}$ with probability $\frac{\lambda-2\epsilon}{d-1}$ and $\widetilde{p}_i = 0$ otherwise. The final distribution is constructed as follow. If $\sum_{i\in[n]} \widetilde{p}_i > 1$ or $\sum_{i\in[n/d]} \widetilde{p}_i < (1-\lambda)$, we simple take $q = p$. Else, we take $p = \widetilde{p}$ and normalize p.

First, we prove it is very unlikely that $\sum_{i\in[n]} \widetilde{p}_i > 1$ or $\sum_{i\in[n/d]} \widetilde{p}_i > (1-\lambda)n/d$. In particular, for $i \in [n]$, let the random variable $Y_i = 1$, if $\widetilde{p}_i > 0$, and $Y_i = 0$ otherwise. Then by Chernoff bound, we have

$$\begin{aligned}\Pr\left[\sum_{i\in[n]} \widetilde{p}_i > 1\right] &= \Pr\left[\sum_{i\in[n]} Y_i > n/d\right] = \Pr\left[\sum_{i\in[n]} Y_i > (1+\epsilon)\sum_{i\in[n]} \mathbb{E}[Y_i]\right] \\ &\le \exp(-\epsilon^2(1-\epsilon)n/3d) \lesssim 1/\operatorname{poly}(n).\end{aligned}$$

where the second step follows from

$$\sum_{i\in[n]} \mathbb{E}[Y_i] = \sum_{i=1}^{\frac{n}{d}} \mathbb{E}[Y_i] + \sum_{i=\frac{n}{d}+1}^{n} \mathbb{E}[Y_i] = (1-\lambda+\epsilon)\cdot\frac{n}{d} + \frac{\lambda-2\epsilon}{d-1}\cdot\frac{(d-1)n}{d} = (1-\epsilon)\cdot\frac{d}{n}.$$

Moreover, we have

$$\begin{aligned}\Pr\left[\sum_{i\in[n/d]} \widetilde{p}_i < (1-\lambda)\right] &= \Pr\left[\sum_{i\in[n/d]} Y_i \le (1-\lambda)n/d\right] \le \Pr\left[\sum_{i\in[n/d]} Y_i \le (1-\epsilon)\sum_{i\in[n/d]} \mathbb{E}[Y_i]\right] \\ &\le \exp(-\epsilon^2(1-\lambda+\epsilon)n/2d) \lesssim 1/\operatorname{poly}(n)\end{aligned}$$

where the second step follows from

$$\sum_{i\in[n/d]} \mathbb{E}[Y_i] = (1-\lambda+\epsilon)\frac{n}{d}.$$

We focus on the case $\sum_{i\in[n]} \widetilde{p}_i \le 1$ and $\sum_{i\in[n/d]} \widetilde{p}_i \ge (1-\lambda)n/d$. One can verify that $\mathrm{TV}(p,q) \le \lambda$. Furthermore, one can show the optimal solution satisfies $\mathrm{OPT}(p) \le (1+3\epsilon)/n$ with high probability, since

$$\begin{aligned}\Pr\left[\sum_{i\in[n]} \widetilde{p}_i \le (1-2\epsilon)n\right] &\le \Pr\left[\sum_{i\in[n]} Y_i \le (1-2\epsilon)n/d\right] = \Pr\left[\sum_{i\in[n]} Y_i \le (1-\epsilon)\sum_{i\in[n]} \mathbb{E}[Y_i]\right] \\ &\le \exp(-\epsilon^2(1-\epsilon)n/2d) \lesssim 1/\operatorname{poly}(n)\end{aligned}$$

and thus $\max_{i\in[n]} p_i \le \frac{1}{1-2\epsilon}\frac{d}{n} \le (1+3\epsilon)\frac{d}{n}$.

Let S_A denote the support of $\widetilde{p}$, i.e., $S_A = \{i : \widetilde{p}_i > 0\}$. It suffices to prove that

$$\mathbb{E}[|N(S_A)|] \le (1-\lambda\exp(-\lambda)+O(\epsilon))n$$

We use X_j to indicate whether server j is connected to some client in S_A, i.e. $X_j = \mathbf{1}\{j \in N(S_A)\}$. Let x_j denote the number of neighbor clients of server j that comes from

$[n/d+1:n]$ and z_j denote the number of neighbor clients of server j that comes from $[n/d]$. We know that $\sum_{j\in[n]} x_j = d\cdot(1-1/d)n = (d-1)n$ and $\sum_{j\in[n]} z_j = d\cdot n/d = n$. We have

$$\begin{aligned}\sum_{j\in[n]} \mathbb{E}[X_j] &= \sum_{j\in[n]} \left(1-\left(1-\frac{\lambda-2\epsilon}{d-1}\right)^{x_j}(\lambda-\epsilon)^{z_j}\right)\\ &\le n - n\left(1-\frac{\lambda-2\epsilon}{d-1}\right)^{\sum_{j\in[n]} x_j/n}(\lambda-\epsilon)^{\sum_{j\in[m]} z_j/n}\\ &= n - n\left(1-\frac{\lambda-2\epsilon}{d-1}\right)^{d-1}(\lambda-\epsilon)\\ &\le (1-\lambda\exp(\lambda)+O(\epsilon)+O(1/d))n\end{aligned}$$

The first step holds since $X_j = 0$ iff none of the x_j clients in $[n/d+1:n]$ and z_j clients in $[n/d]$ belongs to S_A. The second step follows from Jensen's inequality and the convexity of function $f(x,z) = \left(1-\frac{\lambda-2\epsilon}{d-1}\right)^x(\lambda-\epsilon)^z$. We conclude the proof here.

■

We next prove Theorem 13, which asserts the NP-hardness of the network design problem.

Proof [Proof of Theorem 13] We reduce from the subset-sum problem. In the subset-sum problem, we are given k numbers $x_1,\dots,x_k$ satisfying $\sum_{i=1}^k x_i = 2$, and the goal is to find a subset $S\subseteq[k]$ such that $\sum_{i\in S} x_i = 1$. The problem is known to be NP-complete. Given a subset-sum instance $x_1,\dots,x_k$, we set the number of client to be $|A| = k$ and the number of server to be $|B| = (d-1)k+2$. Note one can make the number of clients equals the number of servers by adding $(d-2)k+2$ "dummy" clients whose load are 0. Define the load distribution $p\in\Delta_k$ to be

$$p_i = \frac{d-1+x_i}{(d-1)k+2},\quad \forall i\in A.$$

Suppose there exists $S\subseteq[k]$ such that $\sum_{i\in S} x_i = 1$. then we have an allocation with the maximum load equals $\frac{1}{(d-1)k+2}$. To see this, let the i-th client put $\frac{1}{(d-1)k+2}$ units of load on server $(i-1)(d-1)+1,\dots,i(d-1)$ and put $\frac{x_i}{(d-1)k+2}$ units of load on server $(d-1)k+1$ if $i\in[S]$; it puts $\frac{x_i}{(d-1)k+2}$ units of load on server $(d-1)k+2$ otherwise. It is easy to see that each machine has load $\frac{1}{(d-1)k+2}$ and each client has d connected servers.

On the other side, suppose one can find an allocation with the maximum load $\frac{1}{(d-1)k+2}$, we proceed to show that one can recover a solution to the subset-sum instance. Note that the total number of servers is $|B| = (d-1)k+2$, thus every server has load $\frac{1}{(d-1)k+2}$ in the allocation. For each client $i\in A$, suppose it connects to d servers $B_{i_1},\dots,B_{i_d}$ and assigns loads $\ell_{i_1}\ge\cdots\ge\ell_{i_d}$. Suppose there exists t $(1\le t\le d-1)$ such that $\ell_{i_t}<1$, then we do the following operations. We take $\ell_{i_t}\leftarrow\frac{1}{(d-1)k+2}$ and $\ell_{i_d}\leftarrow\ell_{i_t}+\ell_{i_d}-\frac{1}{(d-1)k+2}$. To keep the maximum load to be $\frac{1}{(d-1)k+2}$, for any client i' that is connected to server B_{i_t}, we delete the connection between B_{it} and client i', re-assign it to server B_{i_d} and transfer all the load. We still maintain a feasible solution, i.e. the maximum load is still $\frac{1}{(d-1)k+2}$ and the number of neighbor servers for each client is still at most d. By doing this for every client, we know

that each client connects to $(d-1)$ servers, for which they put $\frac{1}{(d-1)k+2}$ units of load; and they put their rest load on one of the two left servers. In particular, the i-th client would put $\frac{x_i}{(d-1)k+2}$ units of load load. Thus, we recover a solution to the subset-sum instance. ■

Appendix D. Missing proof from Section 4

Let $\kappa > 0$, the follow Lemma states that each client is connected with at least $(1-\kappa)d$ servers, we omit the proof as it directly follows from the Chernoff bound.

Lemma 29 *Suppose $d \geq \Omega(\log(d)/\kappa^2)$ and we construct the graph with* Randomize greedy*, then with probability $1 - 1/\operatorname{poly}(n)$, $|N(i)| \geq (1-\kappa)d$ holds for every client $i \in [n]$,*

Next, we prove Lemma 15, which asserts that each client is connected to at least $(1/2 - \kappa)d$ clients of low load score, w.h.p.

Proof [Proof of Lemma 15] Fixed a client $i \in [n]$, we prove that $|N(i) \cap B_{\mathsf{light}}| \geq (\frac{1}{2} - \kappa)d$ holds with probability $1 - 1/\operatorname{poly}(n)$, the Lemma is then followed by an union bound. For each server $j \in B$, define

$$\widetilde{s}_j := \sum_{i' \in [n], i' \neq i} \frac{np_{i'}}{d} \mathbf{1}\{j \in N(i')\}.$$

We know that

$$\begin{aligned}\sum_{j\in[n]} \widetilde{s}_j < \sum_{j\in[n]} s_j &= \sum_{j\in[n]} \sum_{i\in[n]} \frac{np_i}{d} \cdot \mathbf{1}\{j \in N(i)\} \\ &= \sum_{i\in[n]} \sum_{j\in[n]} \frac{np_i}{d} \cdot \mathbf{1}\{j \in N(i)\} \leq n \sum_{i\in[n]} p_i = n,\end{aligned}$$

Thus, there are at least $n/2$ servers $j \in [n]$ such that $\widetilde{q}_j \leq 2$, we use $\widetilde{B}_{\mathsf{light}}$ to denote this set. For any $j \in N(i)$, we have

$$q_j = \widetilde{q}_j + \frac{np_i}{d}\mathbf{1}\{j \in N(i)\} = \widetilde{q}_j + \frac{np_i}{d} \leq \widetilde{q}_i + D,$$

Hence, it suffices to prove that $|\widetilde{B}_{\mathsf{light}} \cap N(i)| \geq (\frac{1}{2} - \kappa)d$ holds with high probability. We consider the randomly assigned servers and greedily assigned servers separately.

For randomly assigned neighbors $N_R(i)$, it follows easily from the Hoeffding bound. In particular, we set the random variable $X_k = 1$ ($k \in [|N_R(i)|]$) if the k-th random assigned neighbor server belongs to $\widetilde{B}_{\mathsf{light}}$; and $X_j = 0$ otherwise. It is easy to see that $\{X_k\}_{k\in[|N_R(i)|]}$ is i.i.d. and $\mathbb{E}[\sum_{k\in[|N_R(i)|]} X_k] \geq \frac{1}{2}|N_R(i)|$. Then by Hoeffding bound, one has

$$\begin{aligned}&\Pr\left[\left|N_R(i) \cap \widetilde{B}_{\mathsf{light}}\right| \leq \frac{1}{2}|N_R(i)| - \frac{1}{2}\kappa d\right] \\ =\ &\Pr\left[\sum_{k\in[|N_R(i)|]} X_k \leq \frac{1}{2}|N_R(i)| - \frac{1}{2}\kappa d\right] \\ \leq\ &\exp\left(-\kappa^2 d/2\right) \leq 1/\operatorname{poly}(n). \end{aligned} \tag{35}$$

For the greedily assigned neighbors $N_G(i)$, one crucial observation is that there are at most two servers in $N_G(i)$ that are also assigned greedily for some other client. Since we assume $d = \Omega(\log n)$ we could simply ignore them, and it is w.l.o.g to assume all servers in $N_G(i)$ are not greedily assigned to other client. For any server $j \in N_G(i)$, define the random variable $Y_j = 1$ if $j \in \widetilde{B}_{\mathsf{light}}$. We still have $\mathbb{E}[Y_j] \leq \frac{1}{2}$ due to the symmetricity of all $j \in [n]$ for random assignment.

Furthermore, we claim that $\{Y_j\}_{j \in N_G(i)}$ are negative associated. To see this, for any server $j \in N_G(i)$ and any client i', define

$$Z_{i',j} = \frac{np_{i'}}{d} \mathbf{1}\{j \in N_R(i')\}.$$

We first prove that $\{Z_{i',j}\}_{i',j}$ is NA. We further define $W_{i',j,k}$ where $k \in [d - |N_G(i')|]$, and $W_{i',j,k} = \frac{np_{i'}}{d}$ if the k-th randomly assigned server of client i is server j. It is clear that $\{W_{i',j,k}\}_{j \in [n]}$ is NA since they are permutations $\left(\frac{np_{i'}}{d}, 0, \ldots, 0\right)$ (see Lemma 25). This implies that $\{W_{i',j,k}\}_{i',j,k}$ is also NA by the closure property of NA (see Lemma 24). Moreover, we have

$$Z_{i',j} = \min\left\{\frac{np_{i'}}{d}, \sum_{k=1}^{d-|N_G(i)|} W_{i',j,k}\right\}$$

Thus $\{Z_{i',j}\}_{i',j}$ are also NA since they are defined on disjoint subset of $\{W_{i',j,k}\}_{i',j,k}$ and it is monotone (see Lemma 24). Finally, observe that

$$Y_j = \begin{cases} 0 & \sum_{i' \in [n], i' \neq i} Z_{i',j} \geq 2 \\ 1 & \text{otherwise} \end{cases}$$

Hence $\{Y_j\}_{j \in N_G(i)}$ are also NA since they are defined on disjoint subset of $\{Z_{i',j}\}_{i',j}$ and the function is monotone decreasing.

Now that we proved $\{Y_j\}_{j \in N_G(i)}$ are NA, by the Hoeffding bound for NA variables, we have

$$\begin{aligned} & \Pr\left[\left|N_G(i) \cap \widetilde{B}_{\mathsf{light}}\right| \leq \frac{1}{2} N_G(i) - \frac{1}{2}\kappa d\right] && (36) \\ = \ & \Pr\left[\sum_{j \in N_G(i)} Y_j \leq \frac{1}{2}|N_R(i)| - \frac{1}{2}\kappa d\right] \\ \leq \ & \exp\left(-\kappa^2 d/2\right) \leq \ 1/\operatorname{poly}(n). && (37) \end{aligned}$$

Combining Eq. (35)(37) and Lemma 29, we conclude the proof. ■