Minimum congestion mapping in a cloud

SIAM J. COMPUT.
Vol. 44, No. 3, pp. 819–843

MINIMUM CONGESTION MAPPING IN A CLOUD*

NIKHIL BANSAL†, KANG-WON LEE‡, VISWANATH NAGARAJAN§, AND MURTAZA ZAFER¶

Abstract. We study a basic resource allocation problem that arises in cloud computing environments. The physical network of the cloud is represented as a graph with vertices representing servers and edges corresponding to communication links. A workload is a set of processes with processing requirements and mutual communication requirements. The workloads arrive and depart over time, and the resource allocator must map each workload upon arrival to the physical network. We consider the objective of minimizing the *congestion*. We show that solving a subproblem (SingleMap) about mapping a *single workload* to the physical graph essentially suffices for solving the general problem. In particular, an α-approximation algorithm for SingleMap gives an $O(\alpha \log nD)$ competitive algorithm for the general problem, where n is the number of nodes in the physical network and D is the maximum to minimum workload duration ratio. We then consider the SingleMap problem for two natural classes of workloads, namely *depth-d trees* and *complete-graph* workloads. For depth-d trees, we give an $n^{O(d)}$ time $O(d^2 \log(nd))$-approximation algorithm based on a strong LP relaxation inspired by the *Sherali–Adams* hierarchy. For complete graphs, we give a polylogarithmic approximation algorithm using Räcke decompositions.

Key words. graph mapping, cloud computing, congestion minimization, approximation algorithms

AMS subject classifications. 68W05, 68W40

DOI. 10.1137/110845239

1. Introduction.

In cloud computing, the underlying resource is a physical network (also called the *substrate*) consisting of servers that are interconnected via communication links. Each server has a processing capacity,[1] and each communication link has a bandwidth capacity. *Workloads* are service demands made to the cloud, modeled as a graph with a set of processes with communication requirements between them. Each workload must be assigned/mapped to some physical resources. The goal of the cloud service provider is to allocate resources to workloads in the best possible way.

*Received by the editors August 22, 2011; accepted for publication (in revised form) April 14, 2015; published electronically June 18, 2015. A preliminary version of this paper appeared as [7]. This research was sponsored in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defense and was accomplished under Agreement W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence, or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. The research was also supported through participation in the Measurement Science for Cloud Computing sponsored by the National Institute of Standards and Technology (NIST) under Agreement 60NANB10D003.
http://www.siam.org/journals/sicomp/44-3/84523.html

†Eindhoven University of Technology, Eindhoven, 5600 MB, The Netherlands (n.bansal@tue.nl). This author's work was partially supported by NWO grant 639.022.211 and by ERC consolidator grant 617951.

‡IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 (kangwon@us.ibm.com).

§Industrial and Operations Engineering Department, University of Michigan, Ann Arbor, MI 48109 (viswa@umich.edu).

¶Nyansa Inc., Palo Alto, CA 94303 (murtaza.zafer@gmail.com). This author's work was done during a stay at IBM's T.J. Watson Research Center.

[1]Our results extend easily to settings with multiple resources such as CPU, memory, disk, etc. See section 6.

The assignment of a workload to the substrate can be viewed as mapping one graph into another. This consists of two aspects: (a) *node-mapping*, the assignment of processes (vertices in the workload graph) to servers, and (b) *path-mapping*, the assignment of each communication request (edges in the workload graph) to a path in the substrate between the respective servers. The load on a substrate node (resp., edge) is the total demand using that node (resp., edge). Ideally, one would prefer that the mapping not cause the load on any node or edge to exceed its capacity. However, if this constraint is enforced strictly, the problem can be shown to be hard to approximate to within any reasonable factor. This holds even for simple workloads such as star-shaped networks for trivial reasons (in particular, due to NP-hardness of the partition problem). So, we relax this requirement and consider the natural and well-studied objective of minimizing the maximum node and edge congestion, where the congestion of a node or an edge is defined as the ratio of its load to its capacity.

We will refer to our problem as GraphMap and the objective as *network congestion*. There are two natural variants of GraphMap: In the offline case, the workloads are all known in advance. In the (harder) online case, the workloads arrive and depart over time, and the existence of a workload is unknown until it arrives. Here we seek an online algorithm that assigns each workload (immediately upon its arrival) to the substrate such that the worst-case network congestion over time is minimized. Many previous papers [27, 26, 11, 19] have proposed heuristics to solve such mapping problems but without any performance guarantees. In this paper we design algorithms with provable guarantees.

Interestingly, even the (seemingly simple) problem of mapping a single workload to the substrate is quite hard in general. For example, several classic and well-studied problems are simple, special cases of mapping a single workload:

- *Balanced separator.* The substrate is a single edge with each node having capacity $n/2$, and H is a graph with n nodes. This reduces to partitioning the vertices of the workload H into two near-balanced parts such that the resulting cut is minimized. This is an extensively studied graph partitioning problem, and the best known approximation ratio (when each part is required to have $\Theta(n)$ vertices) is $O(\sqrt{\log n})$ [3].
- *Cut-width.* The substrate is a line on n vertices with equal capacity edges and node capacities 1, and H is an n-node graph. Minimizing edge congestion subject to node capacities reduces to finding an ordering $v_1, \ldots, v_n$ of the vertices in the workload H such that the value of the cut $\max_{i=1}^{n} \delta_H(v_1, \ldots, v_i)$ is minimized. Here, for any subset S of vertices, $\delta_H(S)$ denotes the number of edges of H with exactly one end-point in S. The best known approximation ratio for cut-width is $O(\log^{3/2} n)$ [17].
- *Min-max k-partitioning.* The substrate is a star with k leaves and equal capacity edges; the node capacity of the center is zero, and each leaf has capacity n/k. This reduces to partitioning the vertices of the workload H into k nearly balanced parts $V_1, \ldots, V_k$ such that $\max_{i=1}^{k} \delta_H(V_i)$ is minimized. This is a basic problem in distributed computing, which has been studied only recently [23, 6], and an $O(\sqrt{\log n \, \log k})$-approximation algorithm is known [6], where each part is required to have $O(n/k)$ vertices.

One of our main results will be that the problem of mapping a *single workload* to the substrate essentially captures the hardness of the online GraphMap problem. More precisely, a cost-aware variant of the single workload mapping problem, which we call SingleMap, suffices to solve both the offline and online GraphMap problem. In addition to this connection, we believe that SingleMap is a natural assignment problem on graphs, applicable in a wider context.

As the SingleMap problem (and hence GraphMap) appears very challenging in full generality, we obtain results for the following natural subclasses of workloads that seem to arise most often in practice:

- *Constant depth trees.*
- *Complete graphs* with uniform demands.

Tree-shaped workloads arise commonly, as they correspond to processes arranged hierarchically. The constant depth corresponds to having a small number of hierarchies. The complete graph workloads represent a clique of processes, all of them communicating uniformly with each other. We require that the processing requirements be identical and also that the communication requirement be identical for every pair of processes.[2]

1.1. Model.

The general mapping problem. The substrate is an undirected graph $G = (V, E)$ with edge-capacities $c : E \to \mathbb{R}_+$ and node-capacities $u : V \to \mathbb{R}_+$. There is a set of workloads that need to be mapped into the substrate. Each workload is an undirected virtual graph $H = (W, F)$ with processing demands $g : W \to \mathbb{R}_+$ and traffic demands $b : F \to \mathbb{R}_+$. A *mapping* of H to the substrate G is specified by a tuple $\langle \pi, \sigma \rangle$, where the following hold:

- $\pi : W \to V$ assigns each process $w \in W$ to some node $\pi(w) \in V$ in the substrate G.
- $\sigma : F \to 2^E$ maps each edge $f = (w_1, w_2) \in F$ to a *path* $\sigma(f)$ in G between nodes $\pi(w_1)$ and $\pi(w_2)$; the $b(f)$ units of traffic between processes w_1 and w_2 are routed along $\sigma(f)$. This is an *unsplittable routing* model. An alternative model (which we do not consider) is splittable routing, where $\sigma(f)$ can be a flow of $b(f)$ units between $\pi(w_1)$ and $\pi(w_2)$.

For each edge $e \in E$ in the physical network let L_e denote the total traffic (from all workloads) routed through edge e; then the congestion of edge e is L_e/c_e. Similarly, for a node $v \in V$ let N_v denote the total processing demands assigned to v; then the congestion of v is N_v/u_v. We will refer to L_e (resp., N_v) as the *load* on edge e (resp., vertex v). We define the *network-congestion* to be the maximum of the edge and node congestions.

Given a set of workloads and the substrate graph, the objective in the GraphMap problem is to map all workloads so as to minimize network-congestion. We give algorithms for both offline and online settings. In the offline setting, the algorithm knows all the workloads in advance before computing the mapping. The more realistic online setting is when workloads arrive (and depart) over time, and the algorithm has to make irrevocable assignments to the workloads upon their arrival. We consider the model of known durations (see, e.g., [5]); i.e., each workload specifies upon arrival the time it will spend in the cloud.

If we consider *splittable routing* in the GraphMap problem, then one can assume (at the loss of a polylogarithmic approximation factor) that the substrate G is always a tree [21, 15]. However, this reduction to trees is not applicable in the unsplittable routing model that we consider. Nevertheless, we make use of Räcke decomposition trees [21, 15] in our algorithm for complete-graph workloads.

[2]Such a restriction on the communication requirement is necessary, as any arbitrary workload H can be modeled as a complete graph with zero requirement for edges not in H.

Single workload mapping problem. This will be an important subroutine for both the offline and online versions of GraphMap. The input to SingleMap consists of the following:

- A single workload represented by an undirected graph $H = (W, F)$ with demands $g : W \to \mathbb{R}_+$ and $b : F \to \mathbb{R}_+$.
- Substrate $G = (V, E)$ with edge-capacities $c : E \to \mathbb{R}_+$ and node-capacities $u : V \to \mathbb{R}_+$.
- Cost functions $\alpha : E \to \mathbb{R}_+$ and $\beta : V \to \mathbb{R}_+$.

A mapping of H into G assigns vertices W to V, and each edge in F to a path in G between the respective end-points (as in the definition of GraphMap). As before, for a given mapping, let L_e and N_v denote the load on edges $e \in E$ and vertices $v \in V$. The mapping is called *valid* if it respects all edge and node capacities, i.e., $L_e \le c_e$ $\forall e \in E$ and $N_v \le u_v$ $\forall v \in V$. The goal is to find a valid mapping of H into G that minimizes the total cost $\sum_{e\in E} \alpha_e \cdot L_e + \sum_{v\in V} \beta_v \cdot N_v$.

Note that the objective in SingleMap is not congestion (as in GraphMap), but the cost of the mapping. Using the multiplicative weights update approach, the total cost in SingleMap will be used to control the congestion in GraphMap.

An algorithm for SingleMap is said to be a (ρ_1, ρ_2) *bicriteria approximation* algorithm (where $\rho_1, \rho_2 \ge 1$) if it produces a solution where (a) the cost is at most ρ_1 times the optimum, and (b) the load on each edge or node is at most ρ_2 times its capacity.

1.2. Our results and techniques. First, we describe the general frameworks for designing both offline and online algorithms for GraphMap, assuming a (ρ_1, ρ_2) bicriteria approximation algorithm for SingleMap. In particular, we proceed as follows:

- For offline GraphMap (where all workloads are given up front) we show an $O\left((\rho_1 + \rho_2) \cdot (\log n / \log\log n)\right)$ approximation algorithm. To obtain this result, we formulate a configuration LP relaxation for GraphMap and solve it approximately using SingleMap as the dual separation routine. The final solution is obtained by applying randomized rounding to the approximate LP solution.
- For online GraphMap we give an $O\left((\rho_1 + \rho_2) \cdot \log(nD)\right)$-competitive algorithm, where D is the ratio of maximum to minimum durations. This result uses multiplicative updates and builds upon the ideas developed previously in the context of online virtual circuit routing [4]. We discuss the differences from [4] in subsection 1.3.

These results appear in sections 2 and 3, respectively. An immediate consequence of this framework is that there is a logarithmic approximation for GraphMap on constant-sized workloads. This follows as the SingleMap problem can be solved optimally by simply enumerating all possibilities for such workloads.

Next, we consider SingleMap for arbitrary sized workloads for the cases of constant depth trees and uniform complete graphs. The following theorem is proved in section 4.

THEOREM 1.1. *There is a randomized* $(2,\, O(d^2 \cdot \log nd))$ *bicriteria approximation algorithm for* SingleMap *on d-depth tree workloads, which runs in time* $n^{O(d)}$. *Here n is the number of vertices in the substrate.*

This implies a polynomial time $O(\log n)$-approximation algorithm for SingleMap when $d = O(1)$, and a quasi-polynomial time, polylogarithmic approximation when d is polylogarithmic in n. This result is based on a strong LP relaxation, which is inspired by the *Sherali–Adams* lift-and-project procedure. It is easy to show that direct LP relaxations [11, 19] with just assignment variables have very large integrality gaps even when the workload is a single edge. The main idea in our LP is to use joint assignment variables with d-tuples and "conditional congestion" constraints. The rounding algorithm uses the tree structure of the workload and proceeds in d iterations, each time assigning vertices of a new level via randomized rounding.

For complete graph workloads, the idea is to solve the problem on a tree substrate and then use Räcke decomposition [21, 15]. However, more care is required in this reduction, since the Räcke tree provides only a splittable routing, and we finally want an unsplittable routing.

THEOREM 1.2. *For* GraphMap *under uniform complete-graph workloads we have the following:*

- *An offline $O(\log^3 n)$-approximation algorithm.*
- *An $O(\log^2 n \log\log n \log(nR))$-competitive randomized online algorithm.*

Here n is the number of vertices in the substrate and R is the time horizon of the online algorithm.

1.3. Related work. Even though the graph mapping problem (GraphMap) is very basic, we are not aware of any previous work on it. This problem has two aspects: first, how the vertices of the workload H should be mapped; second, given a mapping of the vertices of H, how to map the edges. Both these issues have been studied separately in previous works. In particular, the quadratic assignment problem [10] is related to the first issue, and the goal there is to find a mapping of *nodes* of one graph into another such that a certain quadratic objective is minimized [16] or maximized [20, 18].

A more closely related problem is the well-known edge-disjoint paths (EDP) problem with congestion [22, 12], which deals with the second issue of how to map edges, although only for single edge workloads. Here, the mapping of the end-points of the workload edge is given, and the goal is to map the edge to a path in the substrate graph to minimize edge congestion. The online version of the EDP problem is referred to as the virtual circuit routing problem [4], which has also been studied extensively.

Another related problem is minimizing congestion for quorum placement on networks [14], for which a polylogarithmic approximation is known. This problem is similar to SingleMap when the workload is a bipartite graph between "client-vertices" and "quorum-vertices." However, there are some crucial differences from SingleMap. In particular, the placement of all client-vertices is fixed, and only the quorum-vertices need to be mapped. Moreover, the demand has a "product multicommodity" structure.

The online framework we present for GraphMap uses ideas from the online virtual circuit routing algorithm [4]. In [4] costs on edges are maintained using multiplicative updates, and each request is routed along a shortest path from its source to its destination. Our framework is a generalization of this result, where requests have more complicated mappings (instead of just a path). Consequently, the subproblem (SingleMap) that we need to solve is also harder, as opposed to shortest-path in [4].

A natural approach to mapping nodes in SingleMap is to consider an LP relaxation similar to those used for quadratic assignment [1, 18]. However, as we show in section 4, such LPs have a large integrality gap for SingleMap. Instead, our result for d-depth tree workloads uses substantially stronger LPs based on the *Sherali–Adams* hierarchy [25]. We are not aware of a more direct approach that yields a polylogarithmic approximation for this problem.

1.4. Preliminaries. We will repeatedly use the following standard probabilistic tail bounds (see, for example, [2]).

LEMMA 1.1. *Let $X_1, \dots, X_n$ be independent random variables taking values in the range $[0, 1]$. Let $X = X_1 + \cdots + X_n$ and $\mu = E[X]$. Then for any $\epsilon > 0$ it holds that $\Pr[X - \mu \geq \epsilon\mu] \leq e^{-c_\epsilon \mu}$, where $c_\epsilon = \epsilon^2/4$ for $\epsilon \leq 2e - 1$, and $c_\epsilon = \epsilon \ln((1 + \epsilon)/e)$ otherwise.*

Another tool that we will need is the "congestion preserving" trees of Räcke, which are useful in reducing a congestion-minimization problem on a general graph to that on a tree. We now describe this formally. Recall that a multicommodity flow on an edge-capacitated graph $G = (V, E)$ with source-sink pairs $\{(s_i, t_i)\}_{i=1}^k$ and demands $\{d_i\}_{i=1}^k$ is a set of flows $\{f_1, \dots, f_k\}$, where f_i carries d_i units of flow from vertices s_i to t_i. The congestion of a multicommodity flow is the maximum ratio (over edges E) of the total flow through an edge to its capacity. A multicommodity flow is said to be feasible if it has congestion at most one.

DEFINITION 1.1. *A tree $T = (V_T, E_T)$ with edge capacities $\{c'_e : e \in E_T\}$ is a β-approximate congestion tree for a graph $G = (V, E)$ with edge capacities $\{c_e : e \in E\}$ if the following hold:*

1. *The vertices V of G are the leaves of T.*
2. *If pairs $\{(s_i, t_i)\}_i$ with demands $\{d_i\}$ have a feasible multicommodity flow in G, then the same pairs (and demands) also have a feasible multicommodity flow in tree T.*
3. *For each pair (s, t) of vertices in V, there is a* flow template *$F_{s,t}$ that defines a unit flow between s and t in G. The flow templates satisfy the following condition: if pairs $\{(s_i, t_i)\}_i$ of leaves in T with demands $\{d_i\}$ have a feasible multicommodity flow in T, then the multicommodity flow in G that routes $d_i \cdot F_{s_i,t_i}$ flow for each i has congestion at most β.*

In a surprising result, Räcke [21] showed that given any undirected graph G there exists a congestion tree T_G where β is polylogarithmic in n. Subsequently, [15] gave a polynomial time algorithm for finding such a tree and also improved β to $O(\log^2 n \log\log n)$. We will refer to such trees as Räcke trees.

2. Offline framework.

We show the following result.

THEOREM 2.1. *For any $\rho_1, \rho_2 \geq 1$ a (ρ_1, ρ_2) bicriteria approximation algorithm for the* SingleMap *problem can be used to obtain an $O\big((\rho_1 + \rho_2) \cdot \frac{\log n}{\log\log n}\big)$ approximation algorithm for the offline* GraphMap *problem.*

The main idea is to solve a *configuration LP* relaxation for GraphMap and then apply randomized rounding. The separation oracle for this LP will be the SingleMap problem.

Let $H_1, \dots, H_k$ denote the workloads to be mapped into substrate $G = (V, E)$ with edge capacities c_e and node capacities u_v. Without loss of generality we assume that the optimum congestion is 1 (the algorithm can do a binary search on the value of the optimum congestion and scale the capacities accordingly). For each $i \in [k] := \{1, 2, \dots, k\}$ let $\mathcal{F}_i$ denote the set of all possible valid mappings of H_i into G such that the load on each edge e (resp., vertex v) is at most c_e (resp., u_v). We define a variable $x_i(\tau)$ for each possible map $\tau \in \mathcal{F}_i$ for H_i. As the optimal solution must use some map from $\mathcal{F}_i$ for each H_i and has overall congestion 1, the following LP is a valid relaxation of GraphMap and has a feasible solution:

$$\begin{aligned}
&\text{minimize } 0\\
&\text{subject to } \sum_{\tau\in\mathcal{F}_i} x_i(\tau) \ge 1 && \forall i\in[k],\\
&\sum_{i=1}^{k}\sum_{\tau\in\mathcal{F}_i} \ell(e,\tau)\cdot x_i(\tau) \le c_e && \forall e\in E,\\
&\sum_{i=1}^{k}\sum_{\tau\in\mathcal{F}_i} \ell(v,\tau)\cdot x_i(\tau) \le u_v && \forall v\in V,\\
&x_i(\tau)\ge 0 && \forall \tau\in\mathcal{F}_i,\ \forall i\in[k].
\end{aligned}$$

Here, for any $i \in [k]$ and $\tau \in \mathcal{F}_i$, $\ell(e,\tau)$ denotes the load on edge $e \in E$ under mapping τ; similarly $\ell(v,\tau)$ denotes the load on vertex $v \in V$.

This LP has an exponential number of variables but only polynomially many constraints, so the natural approach to solving it is via the dual:

$$\begin{aligned}
&\text{maximize } \sum_{i=1}^{k} z_i - \sum_{e\in E} c_e\cdot\alpha_e - \sum_{v\in V} u_v\cdot\beta_v\\
&\text{subject to } \sum_{e\in E}\ell(e,\tau)\cdot\alpha_e + \sum_{v\in V}\ell(v,\tau)\cdot\beta_v \ge z_i && \forall \tau\in\mathcal{F}_i,\ i\in[k],\\
&z_i,\alpha_e,\beta_v \ge 0 && \forall\, i\in[k],\ e\in E,\ v\in V.
\end{aligned}$$

Observe that given the values for (z,α,β) the dual separation problem is precisely SingleMap for each of $\{H_i\}_{i=1}^k$ with capacities c, u and costs α, β. Since we have a (ρ_1, ρ_2) bicriteria approximation algorithm for SingleMap, we can solve the dual LP approximately using the ellipsoid algorithm. By standard LP duality arguments, this gives a primal solution $\{y_i(\tau) : i \in [k], \tau \in \widetilde{\mathcal{F}}_i\}$, where the following hold:

- For each map in $\widetilde{\mathcal{F}}_i$, the load on each edge e (resp., vertex v) is at most $\rho_2 \cdot c_e$ (resp., $\rho_2 \cdot u_v$).
- For all $e \in E$, $\sum_{i=1}^k \sum_{\tau\in\mathcal{F}_i} \ell(e,\tau)\cdot y_i(\tau) \le \rho_1 \cdot c_e$.
- For all $v \in V$, $\sum_{i=1}^k \sum_{\tau\in\mathcal{F}_i} \ell(v,\tau)\cdot y_i(\tau) \le \rho_1 \cdot u_v$.
- Each $\widetilde{\mathcal{F}}_i$ has polynomial size.

Given a primal solution with these properties, the algorithm now chooses a mapping for each workload H_i by picking $\tau \in \widetilde{\mathcal{F}}_i$ independently with probability $y_i(\tau)$. Using Lemma 1.1, it follows that the total load on any edge or vertex is $O((\rho_1 + \rho_2) \cdot (\log n/ \log\log n))$ times its capacity with high probability, which implies the result.

3. Online framework.

In this section we show the following result.

THEOREM 3.1. *Given a (ρ_1, ρ_2) bicriteria approximation algorithm for* SingleMap*, there is an $O((\rho_1 + \rho_2) \log(nD))$-competitive online algorithm for* GraphMap *with known durations. Here n is the number of vertices in the substrate graph, and D is the maximum duration of any workload.*

Above, we assume that each duration is an integer value greater than or equal to one. Using standard arguments, the term D can be replaced with the ratio of maximum to minimum durations.

The algorithm is similar to the online algorithm for virtual circuit routing [4, 5, 9]. The idea is that at each time, the algorithm maintains a cost on the edges and nodes that is an exponentially increasing function of their load. Upon the arrival of a workload, the solution of a SingleMap instance with these costs determines where this workload will be placed. Since the highly loaded edges/nodes are severely penalized, the SingleMap solution will prefer edges/nodes with low load.

Notation. Let $H_1, H_2, \dots, H_k$ denote the workloads in the order in which they arrive; we use i to index the workloads. Each H_i appears at time s_i with a specified duration t_i, which means that H_i stays in the cloud from time s_i to s_i+t_i. We assume that the duration t_i becomes known when H_i arrives at time s_i. Note that the arrival times' s_i's are nondecreasing. We assume that all times and durations are integral and $\max_i t_i \le D$. Also, given a SingleMap algorithm as a black-box, our algorithm for GraphMap will treat edges and vertices identically, and hence we will use the term element to refer to either an edge or a vertex of G. The set of elements will be denoted by U, and c_e will denote the capacity of element $e \in U$. For each workload H_i, let $\mathcal{F}_i$ denote the set of all possible mappings of H_i into G. (Note that this definition differs from the set $\mathcal{F}_i$ used in section 2.) Then for each $i \in [k]$ the algorithm finds a map $\tau_i \in \mathcal{F}_i$, and workload H_i is assigned to G using this map during the interval $[s_i,\, s_i + t_i]$. The network congestion is the maximum congestion over all elements $e \in U$ and over all times h; i.e.,

$$\max_{e\in U}\ \max_{h\ge 0}\ \frac{\sum_{i:s_i\le h\le s_i+t_i}\ \tau_i(e)}{c_e},$$

where $\tau_i(e)$ is the load on element e due to map τ_i for workload H_i.

The SingleMap problem can be restated in the above notation: given a workload H_i, *costs* $\alpha : U \to \mathbb{R}_+$, and capacities $c : U \to \mathbb{R}_+$, find a feasible map $\tau \in \mathcal{F}_i$ minimizing $\sum_{e\in U} \alpha_e \cdot \tau(e)$ such that $\tau(e) \le c_e\ \forall\, e \in U$. As previously, we assume a (ρ_1, ρ_2) bicriteria approximation algorithm for SingleMap.

3.1. Algorithm. In the description below we assume that the optimum offline solution has congestion at most 1. This assumption can be removed by standard doubling techniques (see, for example, [9, Theorem 12.5]), where the online algorithm maintains an upper bound Λ on the optimal congestion thus far. For each $i \in [k]$ let $\tau_i^* \in \mathcal{F}_i$ denote the map in some fixed optimal *offline* solution.

Let $\gamma \in (0,1)$ be a constant to be fixed later. For any $i \ge 1$ let $\ell_i(e,h)$ denote the load of element e at time h, induced by requests $H_1, \dots, H_{i-1}$. Formally,

$$\ell_i(e,h) = \sum_{j=1}^{i-1} \tau_j(e) \cdot \mathbb{I}(h \in [s_j, s_j + t_j]),$$

where $\mathbb{I}(h \in [s_j, s_j + t_j])$ is an indicator 0-1 variable representing whether $s_j \le h \le s_j + t_j$.

Upon the arrival of workload H_i at time s_i, the algorithm does the following:

1. Set costs $\alpha_e := \frac{\gamma}{\rho_2\, c_e} \cdot \sum_{h=s_i}^{s_i+t_i} \exp\left(\frac{\gamma \ell_i(e,h)}{\rho_2 \cdot c_e}\right)\ \forall\, e \in U$.
2. Run the (ρ_1, ρ_2) bicriteria approximation algorithm for SingleMap on instance $\langle H_i, \alpha, c\rangle$ to obtain $\tau_i \in \mathcal{F}_i$ and assign workload H_i according to τ_i during the time interval $[s_i,\, s_i + t_i]$.
3. Update $\ell_{i+1}(e,h) \leftarrow \ell_i(e,h) + \tau_i(e)\ \forall\, e \in U$ and $s_i \le h \le s_i + t_i$.

Note that the above updates to the variables ℓ are consistent with their definition.

3.2. Analysis. We will show that this algorithm is $O((\rho_1 + \rho_2)\log(D|U|))$-competitive. We begin with a simple claim.

CLAIM 3.1. *For any $i \in [k]$ and $\tau \in \mathcal{F}_i$ with $\tau(e) \le \rho_2 \cdot c_e\ \forall\, e \in U$, we have*

$$\sum_{e\in U} \alpha_e \cdot \tau(e) \le \sum_{e\in U} \sum_{h=s_i}^{s_i+t_i} \exp\left(\frac{\gamma \cdot \ell_i(e,h)}{\rho_2 c_e}\right)\left[\exp\left(\frac{\gamma\cdot\tau(e)}{\rho_2 c_e}\right) - 1\right] \le 2\sum_{e\in U}\alpha_e \cdot \tau(e).$$

Proof. For all $x \in [0,1]$ and $\gamma \in (0,1)$ we have $\exp(\gamma x) - 1 \in [\gamma x, 2\gamma x]$. Consider any $e \in U$. As $\tau(e) \in [0,\, \rho_2 \cdot c_e]$, setting $x = \tau(e)/(\rho_2 c_e)$ above,

$$\frac{\gamma\cdot\tau(e)}{\rho_2 c_e} \le \exp\left(\frac{\gamma\cdot\tau(e)}{\rho_2 c_e}\right) - 1 \le 2\frac{\gamma\cdot\tau(e)}{\rho_2 c_e}.$$

The claim now follows by the definition of costs $\alpha_e = \frac{\gamma}{\rho_2\, c_e}\cdot\sum_{h=s_i}^{s_i+t_i}\exp(\gamma\cdot\ell_i(e,h)/\rho_2\cdot c_e)$ and summing over e. ∎

For each $e \in U$ and time $h \ge 0$, let $\overline{\ell}(e,h) = \max_i \ell_i(e,h) = \ell_{k+1}(e,h)$ denote the observed load on element e at time h. Clearly, the objective value of the online algorithm is $\max_{e\in U}\max_{h\ge 0}\overline{\ell}(e,h)/c_e$, which we wish to bound. To this end, for any integer $j \ge 1$ define

$$L_j = \sum_{e\in U}\ \sum_{h=(j-1)\cdot D}^{j\cdot D} \exp\left(\frac{\gamma\cdot\overline{\ell}(e,h)}{\rho_2\, c_e}\right).$$

We will show the next result.

LEMMA 3.1. *Setting $\gamma := \min\{\frac{\rho_2}{6\rho_1}, 1\}$, for each $j \ge 1$, we have $L_j \le 6\cdot D\,|U|$.*

Before we prove Lemma 3.1, we note that this already implies our main result, Theorem 3.1. In particular, $\forall\, e \in U$ and $h \ge 0$, taking logarithms,

$$\overline{\ell}(e,h) \le \frac{1}{\gamma}\ln(6D\,|U|)\cdot\rho_2\, c_e \le \frac{\rho_2\,\ln(6D\,|U|)}{\gamma}\cdot c_e \le \max\{\rho_2,\, 6\rho_1\}\,\ln(6D|U|)c_e$$

by the definition of γ.

Proof of Lemma 3.1. The proof proceeds by induction on j. Define $L_0 = 0$ for the base case. Consider now any $j \ge 1$, assuming inductively that $L_{j-1} \le 6\cdot D\,|U|$. Let $R = \{r, r+1, \dots, t\}$ denote the indices of workloads that are released in the interval $[(j-2)D, jD]$. For any index $i \in R \cup \{t+1\}$ define

$$A_i = \sum_{e\in U}\ \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\gamma\cdot\frac{\ell_i(e,h)}{\rho_2\, c_e}\right).$$

CLAIM 3.2. $A_r \le 2D\,|U| + L_{j-1}$.

Proof. By the choice of R, $s_{r-1} < (j-2)D$. As D is the maximum duration, $s_{r-1} + t_{r-1} < (j-1)D$, and hence $\ell_r(e,h) = 0\ \forall\, e \in U$ and $h \ge (j-1)D$. Thus,

$$\begin{aligned} A_r &= \sum_{e\in U}\left(\sum_{h=(j-2)D}^{(j-1)D}\exp\left(\frac{\gamma\cdot\ell_r(e,h)}{\rho_2\, c_e}\right) + \sum_{h=(j-1)D}^{(j+1)D}\exp(0)\right)\\ &\le \sum_{e\in U}\left(\sum_{h=(j-2)D}^{(j-1)D}\exp\left(\frac{\gamma\cdot\overline{\ell}(e,h)}{\rho_2\, c_e}\right) + 2D\right)\\ &= L_{j-1} + 2D\,|U|. \qquad ∎\end{aligned}$$

For convenience, for any $i \in [k]$, $e \in U$, and $h \geq 0$, let us define $\tau_i^*(e,h) = \tau_i^*(e)$ if $s_i \leq h \leq s_i + t_i$, and 0 otherwise. Also define $\tau_i(e,h)$ similarly. Note that $\ell_i(e,h) = \sum_{p=1}^{i-1} \tau_p(e,h)$ for any $i \in [k]$, $e \in U$, and $h \geq 0$.

CLAIM 3.3. *For any $i \in R$, we have*

$$A_{i+1} - A_i \;\leq\; 2\rho_1\gamma \cdot \sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_{t+1}(e,h)}{\rho_2\, c_e}\right) \cdot \frac{\tau_i^*(e,h)}{\rho_2\, c_e}.$$

Proof. By definition,

$$A_{i+1} - A_i = \sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_i(e,h)}{\rho_2\, c_e}\right) \left[\exp\left(\frac{\gamma \cdot \tau_i(e,h)}{\rho_2\, c_e}\right) - 1\right].$$

Recall that τ_i is a (ρ_1, ρ_2) bicriteria approximation for a SingleMap instance with capacities c; so $\tau_i(e,h) \leq \tau_i(e) \leq \rho_2 \cdot c_e$. Now we can use (see Claim 3.1) that $\exp\left(\gamma\, \frac{\tau_i(e,h)}{\rho_2\, c_e}\right) - 1 \leq 2\gamma \cdot \frac{\tau_i(e,h)}{\rho_2\, c_e}$, and hence $A_{i+1} - A_i$ is at most

$$\begin{aligned} &\sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_i(e,h)}{\rho_2\, c_e}\right) \cdot \frac{2\gamma \cdot \tau_i(e,h)}{\rho_2\, c_e} \\ &\qquad = \sum_{e\in U} \sum_{h=s_i}^{s_i+t_i} \exp\left(\frac{\gamma \cdot \ell_i(e,h)}{\rho_2\, c_e}\right) \cdot \frac{2\gamma \cdot \tau_i(e)}{\rho_2\, c_e} \\ &\qquad = 2 \sum_{e\in U} \alpha_e \cdot \tau_i(e). \qquad (1) \end{aligned}$$

The first equality is by definition of $\tau_i(e,h)$, and the second is by the definition of the α_e's.

As τ_i^* is also a candidate feasible solution to this instance (recall the assumption that optimal congestion is at most one) and τ_i is a (ρ_1, ρ_2)-approximate solution to this SingleMap instance, we have

$$\sum_{e\in U} \alpha_e \cdot \tau_i(e) \leq \rho_1 \cdot \sum_{e\in U} \alpha_e \cdot \tau_i^*(e). \qquad (2)$$

Moreover, since $\ell_i(e,h) \leq \ell_{t+1}(e,h)$ and $(j-2)D \leq s_i \leq s_i + t_i \leq (j+1)D$ for any $i \in R$, we have

$$\alpha_e \leq \frac{\gamma}{\rho_2\, c_e} \cdot \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_{t+1}(e,h)}{\rho_2\, c_e}\right) \quad \forall e \in U.$$

Combining this with (1) and (2), we obtain

$$A_{i+1} - A_i \;\leq\; 2\rho_1\gamma \cdot \sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_{t+1}(e,h)}{\rho_2\, c_e}\right) \cdot \frac{\tau_i^*(e,h)}{\rho_2\, c_e},$$

which proves the claim. ◻

Summing the inequality in Claim 3.3 over all $i \in R$, we can upper bound $A_{t+1} - A_r$ as

$$\leq 2\rho_1\gamma \cdot \sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_{t+1}(e,h)}{\rho_2\, c_e}\right) \cdot \left(\sum_{i\in R} \frac{\tau_i^*(e,h)}{\rho_2\, c_e}\right)$$

$$\leq \frac{2\rho_1\gamma}{\rho_2} \cdot \sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_{t+1}(e,h)}{\rho_2\, c_e}\right). \tag{3}$$

The second inequality uses our assumption that the optimum congestion is at most 1; i.e., $\sum_i \tau_i^*(e,h) \leq c_e$ for all elements e and time h.

As $\gamma := \min\{\frac{\rho_2}{6\rho_1}, 1\}$, using the definition of A_{t+1} in (3), we have

$$A_{t+1} - A_r \leq \frac{2\rho_1\gamma}{\rho_2} \cdot A_{t+1} \leq \frac{1}{3} A_{t+1},$$

which implies that $A_{t+1} \leq 1.5 \cdot A_r$. Together with Claim 3.2, this implies that

$$A_{t+1} \leq 3D\,|U| + 1.5\, L_{j-1}.$$

On the other hand, $A_{t+1} \geq L_{j-1} + L_j$. This follows because, by definition of R, workload $t+1$ arrives after time jD, i.e., $s_{t+1} > jD$, and so $\forall\, e \in U$ and $h \leq jD$ we have $\ell_{t+1}(e,h) = \overline{\ell}(e,h)$. Thus, $L_{j-1} + L_j \;\leq\; A_{t+1} \;\leq\; 3D\,|U| + 1.5 \cdot L_{j-1}$, and hence

$$L_j \leq 3D\,|U| + \frac{L_{j-1}}{2}.$$

As $L_{j-1} \leq 6D\,|U|$ by the inductive hypothesis, this proves Lemma 3.1. ∎

4. Single workload mapping for *d*-depth tree workloads.

In this section we prove Theorem 1.1. As mentioned previously, our result is based on an LP relaxation inspired by the Sherali–Adams hierarchy. It is instructive to see why simpler approaches do not seem to work. Clearly, an LP relaxation just based on assignment variables $y_{p,v}$ (which indicate that node p is mapped to vertex v) is very weak, as it cannot capture the pairwise traffic constraints. However, it turns out that even a quadratic assignment-type LP with variables y_{p_i,v_i,p_j,v_j} (representing whether p_i mapped to v_i *and* p_j mapped to v_j) is also very weak, as shown next.

In particular, consider a star workload with center r and n leaves $\ell_1, \ldots, \ell_n$ with unit traffic and processing demands. The substrate consists of n disjoint edges $\{(a_i, b_i)\}_{i=1}^n$ each of capacity one; each vertex also has capacity one. All costs are zero, so this is just a feasibility question.

Clearly, any integral mapping must violate the capacity of some edge by a factor of n. However, it turns out that there is a feasible solution for quadratic assignment LPs [1] that satisfies all the capacities. Below we use $[n] := \{1, 2, \ldots, n\}$. The fractional solution y is as follows:

$$y(r,v) = \begin{cases} \frac{1}{n} & \text{if } v \in \{a_i\}_{i=1}^n, \\ 0 & \text{otherwise,} \end{cases} \qquad y(\ell_j, v) = \begin{cases} \frac{1}{n} & \text{if } v \in \{b_i\}_{i=1}^n, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall j \in [n].$$

For each $i, j \in [n]$ we have

$$y(r, a_i, \ell_j, v) = \begin{cases} \frac{1}{n} & \text{if } v = b_i, \\ 0 & \text{otherwise.} \end{cases}$$

Basically this solution is a convex combination of the n integral solutions, where for each $i \in [n]$, r maps to a_i and all the leaves $\{\ell_j\}_{j=1}^n$ map to b_i. This LP solution is feasible, as the total usage of each edge $\{(a_i, b_i)\}_{i=1}^n$ is one, so edge capacities are satisfied. Similarly the total usage of each vertex is also at most one.

The trouble with this LP is that it fails to capture the fact that, when the center is mapped to some vertex s, the traffic from *all* leaves must come to s. To get around this problem, we will add additional constraints that we call *conditional congestion constraints*. Roughly speaking, they ensure that, conditional on the center being mapped to some vertex s, the total congestion induced by all edges remains at most one. These are formally described later.

Before describing our LP based algorithm for d-level tree workloads, we describe a simpler combinatorial algorithm for star workloads with uniform demands. This is useful, as such workloads are likely to appear frequently in practice and combinatorial algorithms are simpler to implement than LP-based approaches. Also, this algorithm explicitly illustrates the problem with the LP described above and motivates the Sherali–Adams approach better. Interestingly, we do not know how to extend this combinatorial algorithm even to trees with depth two.

4.1. Exact algorithm for uniform star workload. Here we consider the special case of a star-shaped workload with uniform demand. Let ℓ denote the number of edges in the star workload and $b \in \mathbb{R}_+$ the demand on each edge. All vertices of the star have unit processing demands. The substrate $G = (V, E)$ is an arbitrary graph.

The algorithm: For each vertex $s \in V$, we do the following: Define a flow network N_s on G with s as source and a new sink vertex t that is connected to all vertices V. Set the capacity of each edge $e \in E$ to be $\lfloor c_e/b \rfloor$, the capacity of each edge (v, t) to be u_v (for $v \in V \setminus \{s\}$), and the capacity of (s, t) to $u_s - 1$. There is a cost of α_e on each edge $e \in E$, and cost of β_v for each edge (v, t).

The network flow instance on N_s involves computing the *minimum cost flow* of ℓ units from s to t, which can be done efficiently [13]. Observe that there is a one-to-one correspondence between feasible solutions to this flow instance N_s and valid mappings of the star-workload where the center is mapped to s. Note that having fixed the center at s, the flow instance N_s captures both node and path mappings. Thus the minimum cost optimum amongst instances $\{N_s : s \in V\}$ yields an optimal solution to SingleMap on uniform star workloads.

The main idea in the above algorithm was to enumerate over the mapping of the center (s), which enabled a reduction to single commodity flow. This approach can be extended to workloads with a constant number of nonleaf vertices, since we can again enumerate over all nonleaves and reduce to single commodity flow. However, extending this idea to even a 2-level tree workload appears problematic since we can no longer perform such an enumeration (if there are superlogarithmic numbers of nonleaf vertices).

4.2. LP-relaxation for *d*-depth tree workload.

Notation. We fix some notation relevant to this section. We use $H = (W, F)$ to denote the workload which is a tree of depth d rooted at some node r. The *level* of a vertex $v \in W$ is the number of edges on the path from v to root r. So the root has level zero. The level of an edge is the maximum of the levels of its end-points. For any $i \in \{0, \ldots, d\}$ we use p_i to refer to some node at level i (p_0 is always the root r). An edge (p_i, p_{i+1}) has demand $b(p_i, p_{i+1})$, and a node p has processing demand $g(p)$. The substrate is an arbitrary graph $G(V, E)$ with edge and vertex capacities c_e and u_v. The costs of the edges and vertices are $\{\alpha_e\}_{e \in E}$ and $\{\beta_v\}_{v \in V}$. For an i-level node

p_i, we use $(p_0, \ldots, p_i)$ to denote the (unique) path in H from the root p_0 to p_i.

The LP relaxation. We describe here the LP relaxation. First, we describe the variables we use. There will be two types of variables, which we call assignment variables and flow variables.

Assignment variables: For every index $i \in \{0, \ldots, d\}$, path $(p_0, p_1, \ldots, p_i)$ in H, and all vertices $v_0, \ldots, v_i \in V$, we introduce a variable:

$$y(p_0, v_0, \ldots, p_i, v_i) \in [0, 1]. \tag{4}$$

In an integral solution, this variable is intended to be 1 if for each $j \in \{0, \ldots, i\}$ vertex p_j is mapped to v_j. It is convenient to view this variable as the probability of the event $\bigwedge_{j=0}^{i}(p_j \text{ is mapped to } v_j)$. Also, we allow only variables where each $p \in W$ is mapped to some $v \in V$ with $g(p) \leq u_v$; we set all other y variable to zero. Formally,

$$y(p_0, v_0, \ldots, p_i, v_i) = 0 \quad \text{if } \exists j \in \{0, \ldots, i\} : g(p_j) > u_{v_j}. \tag{5}$$

Flow variables: For every path $(p_0, p_1, \ldots, p_i)$ in H with $i \geq 1$ and collection of vertices $v_0, \ldots, v_i \in V$ we will define a network flow instance. This instance will be denoted by $\mathcal{F}(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i)$ and is supposed to correspond to the mapping of edge (p_{i-1}, p_i) under the event that "p_j is mapped to v_j for each $j \in \{0, \ldots, i\}$." For each edge $e = \{v, w\} \in E$ there are two variables in this flow instance:

$$\mathcal{F}_{vw}(p_0, v_0, \ldots, p_i, v_i) \geq 0 \quad \text{and} \quad \mathcal{F}_{wv}(p_0, v_0, \ldots, p_i, v_i) \geq 0. \tag{6}$$

These two variables denote the flow from v to w and w to v (respectively).

The underlying network $N(p_{i-1}, v_{i-1}, p_i, v_i)$ in this flow instance is the substrate graph G restricted to edges of capacity at least $b(p_{i-1}, p_i)$. So the flow on edges of $G \setminus N(p_{i-1}, v_{i-1}, p_i, v_i)$ is *fixed to zero*; i.e.,

$$\mathcal{F}_{vw}(p_0, v_0, \ldots, p_i, v_i) = \mathcal{F}_{wv}(p_0, v_0, \ldots, p_i, v_i) = 0 \quad \text{if } b(p_{i-1}, p_i) > c_{vw}. \tag{7}$$

The network $N(p_{i-1}, v_{i-1}, p_i, v_i)$ has source-vertex v_{i-1} and sink-vertex v_i. The variables satisfy flow-conservation constraints at all vertices except the source and sink:

$$\sum_{w \in V} \mathcal{F}_{vw}(p_0, v_0, \ldots, p_i, v_i) - \sum_{w \in V} \mathcal{F}_{wv}(p_0, v_0, \ldots, p_i, v_i) = 0 \quad \forall v \in V \setminus \{v_{i-1}, v_i\}. \tag{8}$$

Finally we ensure that $y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i) \cdot b(p_{i-1}, p_i)$ units of flow are sent from v_{i-1} to v_i:

$$\begin{aligned}\sum_{w \in V} \mathcal{F}_{wv_i}(p_0, v_0, \ldots, p_i, v_i) - \sum_{w \in V} \mathcal{F}_{v_i w}(p_0, v_0, \ldots, p_i, v_i)& \\ = y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i) \cdot b(p_{i-1}, p_i)&.\end{aligned} \tag{9}$$

By the integrality of the flow polytope, we have the next result.

CLAIM 4.1. *There is an efficiently computable probability distribution of $v_{i-1} - v_i$ paths in network $N(p_{i-1}, v_{i-1}, p_i, v_i)$ where the marginal probability of each edge (v, w) is*

$$\frac{\mathcal{F}_{vw}(p_0, v_0, \ldots, p_i, v_i)}{y(p_0, v_0, \ldots, p_i, v_i) \cdot b(p_{i-1}, p_i)}.$$

For notational convenience we also define the following variables:

$$\mathcal{F}_e(p_0, v_0, \ldots, p_i, v_i) = \mathcal{F}_{vw}(p_0, v_0, \ldots, p_i, v_i) + \mathcal{F}_{wv}(p_0, v_0, \ldots, p_i, v_i) \quad \forall e = \{v, w\} \in E. \tag{10}$$

These correspond to the net flow (in either direction) through each edge in E in the network flow instance $\mathcal{F}(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i)$.

We impose three types of constraints.

Consistency constraints: Since we intend the y variables to model probabilities, we impose the following natural consistency constraints:

1. For all paths $(p_0, p_1, \ldots, p_i)$ in H and $v_0, \ldots, v_{i-1} \in V$,

$$\sum_{v \in V} y(p_0, v_0, \ldots, p_i, v) = y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}). \tag{11}$$

   This can be viewed as saying that

$$\left\{ \frac{y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v)}{y(p_0, v_0, \ldots, p_{i-1}, v_{i-1})} \right\}_{v \in V}$$

   defines a valid probability distribution for mapping p_i to V conditional upon $p_0, \ldots, p_{i-1}$ being mapped to $v_0, \ldots, v_{i-1}$, respectively.

2. As the root must be assigned somewhere, we have

$$\sum_{v \in V} y(p_0, v) = 1. \tag{12}$$

Together, (11) and (12) imply that every path $(p_0, p_1, \ldots, p_i)$ in H is mapped somewhere; i.e.,

$$\sum_{v_0, \ldots, v_i \in V} y(p_0, v_0, \ldots, p_i, v_i) = 1.$$

Global congestion constraints: These ensure that the load of each edge or vertex in G is at most its capacity. For each edge $e \in E$ we have

$$\sum_{(p_{i-1}, p_i) \in F} \; \sum_{v_0, \ldots, v_i} \mathcal{F}_e(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i) \;\le\; c_e. \tag{13}$$

Above, for any edge $(p_{i-1}, p_i) \in F$, vertices $p_0, \ldots, p_i$ denote its (unique) path in H from the root. Note that the left-hand side is precisely the total fractional load on edge e due to all edges (p_{i-1}, p_i) in the workload.

Similarly, for each vertex $v \in V$ we have

$$\sum_{p_i \in W} g(p_i) \cdot \sum_{v_0, \ldots, v_{i-1}} y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v) \;\le\; u_v. \tag{14}$$

Again, for any vertex $p_i \in W$, vertices $p_0, \ldots, p_i$ denote its (unique) path in H from the root.

Conditional congestion constraints: These final types of constraints are perhaps the least natural, but they are critical to strengthening the LP.

For each index $i \ge 0$, each path $(p_0, \ldots, p_i)$ in H, each choice of vertices $v_0, \ldots, v_i \in V$, and each edge $e \in E$, we add the following constraint:[3]

[3]Here we set variables $\mathcal{F}_e(p_0, v) = 0 \; \forall e \in E$ and $v \in V$ (recall that such flow variables are not defined in the LP).

$$\mathcal{F}_e(p_0, v_0, \dots, p_i, v_i) + \sum_{j \geq i+1} \sum_{p_{i+1}, v_{i+1}, \dots, p_j, v_j} \mathcal{F}_e(p_0, v_0, \dots, p_i, v_i, \dots p_j, v_j) \leq c_e \cdot y(p_0, v_0, \dots, p_i, v_i). \tag{15}$$

This constraint is similar to the global edge congestion constraint, except that we condition on the event that $p_0, \dots, p_i$ are mapped to $v_0, \dots, v_i$, respectively. That is, conditional on $p_0, \dots, p_i$ being mapped to $v_0, \dots, v_i$, the total load on e due to mapping all edges in the subtree rooted at p_i plus the edge (p_{i-1}, p_i) must be no more than c_e.

Similarly, for each vertex $v \in V$, index $i \geq 0$, path $(p_0, \dots, p_i)$ in H, and all vertices $v_0, \dots, v_i \in V$, we add the following constraint:

$$I[v = v_i] \cdot g(p_i) \cdot y(p_0, v_0, \dots, p_i, v_i) + \sum_{j \geq i+1} \sum_{p_{i+1}, v_{i+1}, \dots, p_j} g(p_j) \cdot y(p_0, v_0, \dots, p_i, v_i, \dots p_j, v) \leq u_v \cdot y(p_0, v_0, \dots, p_i, v_i). \tag{16}$$

Above, $I[v = v_i]$ equals one if $v = v_i$ and is zero otherwise. That is, conditional on $p_0, \dots, p_i$ being mapped to $v_0, \dots, v_i$, the load on v due to nodes in the subtree rooted at p_i must be no more than u_v.

Objective: The objective is to minimize

$$\sum_{e \in E} \alpha_e \cdot \sum_{(p_{i-1}, p_i) \in F} \sum_{v_0, \dots, v_i} \mathcal{F}_e(p_0, v_0, \dots, p_{i-1}, v_{i-1}, p_i, v_i) + \sum_{v \in V} \beta_v \cdot \sum_{p_i \in W} g(p_i) \cdot \sum_{v_0, \dots, v_{i-1}} y(p_0, v_0, \dots, p_{i-1}, v_{i-1}, p_i, v). \tag{17}$$

Here, the first term measures the cost due to the fractional load on the edges, and the second term measures the cost due to fractional load on the vertices.

This completes the description of the LP. Observe that the total number of variables and constraints is $n^{O(d)}$, which is polynomial for constant d. Hence this LP can be solved exactly in $n^{O(d)}$ time. Moreover, this LP is a valid relaxation of the SingleMap problem with d-depth tree workloads, as shown next.

CLAIM 4.2. *The optimal value of the LP with constraints* (4)–(16) *and objective* (17) *is at most that of the* SingleMap *instance.*

Proof. Consider any feasible mapping $\langle \pi, \sigma \rangle$ of the given SingleMap instance. Here $\pi : W \to V$ maps each vertex $p_i \in W$ (of the workload) to vertex $\pi(p_i) \in V$ (of the substrate). And $\sigma : F \to 2^E$ maps each edge $(p_i, p_{i+1}) \in F$ to a path in G from $\pi(p_i)$ to $\pi(p_{i+1})$. We will show that this corresponds to a feasible solution to the above LP, with objective (17) at most the SingleMap objective of $\langle \pi, \sigma \rangle$.

We set the assignment variables as follows. For each $i \geq 0$ and path $(p_0, p_1, \dots, p_i)$ in H, set

$$y(p_0, \pi(p_0), p_1, \pi(p_1) \dots, p_i, \pi(p_i)) = 1.$$

The flow variables are set as follows. For each $i \geq 1$ and path $(p_0, p_1, \dots, p_i)$ in H, set

$$\mathcal{F}_{vw}(p_0, \pi(p_0), p_1, \pi(p_1) \dots, p_i, \pi(p_i)) = b(p_{i-1}, p_i) \quad \forall (v, w) \in \sigma(p_{i-1}, p_i).$$

Above, all edges in the $v_{i-1} - v_i$ path $\sigma(p_{i-1}, p_i)$ are directed from v_{i-1} to v_i. All other variables are set to zero.

Constraints (4), (7), (9), (11), and (12) hold by definition of the assignment and flow variables. All the other constraints can be easily verified to hold since mapping $\langle \pi, \sigma \rangle$ is feasible for the SingleMap instance. Similarly, the objective (17) is exactly the SingleMap objective of $\langle \pi, \sigma \rangle$. $\square$

4.3. The rounding algorithm. We round the optimal LP solution in d phases, where in the ith phase we fix the mapping of all level i vertices in H.

Vertex mapping: The algorithm incrementally constructs a mapping $\tau : W \to V$ as follows:

```
1. Set $\tau(p_0) \leftarrow v$ with probability $y(p_0, v)$. This fixes the mapping of the root.
2. For each $i \in \{1, \dots, d\}$ do:
   For each vertex $p_i$ at level $i$,
      • Let $(p_0, \dots, p_{i-1}, p_i)$ denote the path from the root to $p_i$.
      • Set $\tau(p_i) \leftarrow v$ independently with probability
```

$$\frac{y(p_0, \tau(p_0), \dots, p_{i-1}, \tau(p_{i-1}), p_i, v)}{y(p_0, \tau(p_0), \dots, p_{i-1}, \tau(p_{i-1}))}. \tag{18}$$

Note that the algorithm is well defined, as at any iteration i the map τ is already known for all vertices at levels up to $i-1$. Also, (18) defines a valid (conditional) probability distribution for mapping p_i, due to LP constraint (11).

Edge mapping: Having obtained the vertex mapping τ above, the map σ from edges of H to paths in G is constructed by randomized rounding. For each edge (p_{i-1}, p_i) in H do:

- Obtain a flow-path decomposition of

$$\left\{ \frac{\mathcal{F}_{vw}(p_0, \tau(p_0), \dots, p_{i-1}, \tau(p_{i-1}), p_i, \tau(p_i))}{b(p_{i-1}, p_i) \cdot y(p_0, \tau(p_0), \dots, p_{i-1}, \tau(p_{i-1}), p_i, \tau(p_i))} \right\}_{v,w}.$$

  By Claim 4.1, this gives a probability distribution on $\tau(p_{i-1})$ to $\tau(p_i)$ paths in $N(p_{i-1}, \tau(p_{i-1}), p_i, \tau(p_i))$.
- Assign edge (p_{i-1}, p_i) to a random $\tau(p_{i-1})$ to $\tau(p_i)$ path chosen according to the above distribution; call this path $\sigma(p_{i-1}, p_i)$, and send $b(p_{i-1}, p_i)$ units of flow along $\sigma(p_{i-1}, p_i)$.

This completes the description of the rounding procedure.

Two simple properties. We note here two useful properties of this procedure:

1. For any path $(p_0, \dots, p_i) \in H$ and vertices $v_0, \dots, v_i \in V$,

$$\Pr[\tau(p_0) = v_0, \dots, \tau(p_i) = v_i] = y(p_0, v_0, \dots, p_i, v_i).$$

2. Similarly, for any edge $e \in E$ and edge $(p_{i-1}, p_i) \in F$ with $(p_0, \dots, p_i)$ being its path from r and $v_0, \dots, v_i \in V$,

$$\begin{aligned} &\Pr[e \in \sigma(p_{i-1}, p_i) \mid \tau(p_0) = v_0, \dots, \tau(p_i) = v_i] \\ &\quad = \frac{\mathcal{F}_e(p_0, v_0, \dots, p_{i-1}, v_{i-1}, p_i, v_i)}{b(p_{i-1}, p_i) \cdot y(p_0, v_0, \dots, p_{i-1}, v_{i-1}, p_i, v_i)}. \end{aligned} \tag{19}$$

4.4. The analysis. We need to show two things: first, that the cost of the mapping is close to optimum; second, that the edge and node congestions are not too high.

CLAIM 4.3. *The expected cost of the algorithm's mapping $\langle \tau, \sigma \rangle$ equals the optimal LP objective.*

This claim, along with the Markov inequality, implies that with probability at least half, the cost of $\langle\tau,\,\sigma\rangle$ is at most twice the LP optimum.

Proof of Claim 4.3. The cost of any mapping $\langle\tau,\,\sigma\rangle$ is

$$\sum_{p\in W} g(p)\cdot\beta_{\tau(p)} + \sum_{(p,q)\in F} b(p,q)\cdot\sum_{e\in\sigma(p,q)}\alpha_e,$$

given by the total of node costs and edge costs.

For any i-level node p_i with $(p_0,\ldots,p_{i-1},p_i)$ as its path from the root and vertices $v_0,\ldots,v_i\in V$, recall that our rounding procedure satisfies $\Pr[\tau(p_0)=v_0,\ldots,\tau(p_i)=v_i]=y(p_0,v_0,\ldots,p_i,v_i)$. So,

$$\Pr[\tau(p_i)=v] \;=\sum_{v_0,\ldots,v_{i-1}} y(p_0,v_0,\ldots,p_{i-1},v_{i-1},p_i,v),$$

and hence the expected node-cost of mapping $\langle\tau,\,\sigma\rangle$ is

$$\sum_{p_i\in W} g(p_i)\sum_v \beta_v\cdot\Pr[\tau(p_i)=v]=\sum_v\beta_v\sum_{p_i\in W} g(p_i)\sum_{v_0,\ldots,v_{i-1}} y(p_0,v_0,\ldots,p_{i-1},v_{i-1},p_i,v),$$

which is exactly the second term in the LP objective (17).

We now compute the expected edge-cost. Consider any edge $(p_{i-1},p_i)\in F$. By (19), and unconditioning over the events $\tau(p_0)=v_0,\ldots,\tau(p_{i-1})=v_{i-1},\ \tau(p_i)=v_i$, we obtain

$$\Pr[\sigma(p_{i-1},p_i)\ni e]=\sum_{v_0,\ldots,v_i}\frac{\mathcal{F}_e(p_0,v_0,\ldots,p_{i-1},v_{i-1},p_i,v_i)}{b(p_{i-1},p_i)}.$$

Thus the expected edge-cost is

$$\begin{aligned}&\sum_{(p_{i-1},p_i)\in F} b(p_{i-1},p_i)\cdot\sum_{e\in E}\alpha_e\cdot\Pr[\sigma(p_{i-1},p_i)\ni e]\\ &\quad=\sum_{e\in E}\alpha_e\sum_{(p_{i-1},p_i)\in F}\ \sum_{v_0,\ldots,v_i}\mathcal{F}_e(p_0,v_0,\ldots,p_{i-1},v_{i-1},p_i,v_i),\end{aligned}$$

which is exactly the first term in the LP objective (17). This implies the claim. ∎

Bounding edge and node congestion. We now bound the edge and node congestion of the mapping produced by our algorithm.

THEOREM 4.1. *With probability at least* $1-1/n^2$*, the maximum node or edge congestion is at most* $O(d^2\log(nd))$.

We describe here the analysis for edge congestion; the analysis for node congestion is essentially identical.

Fix an edge $e\in E$ in the substrate. For each level i edge $(p_{i-1},p_i)\in F$ in the workload, the load assigned by the LP solution to e is

$$\sum_{v_0,\ldots,v_i}\mathcal{F}_e(p_0,v_0,\ldots,p_{i-1},v_{i-1},p_i,v_i).$$

We will be interested in how this load evolves as the rounding proceeds on each level of nodes in W.

For $\ell\in\{0,\ldots,d\}$, let $\tau^{(\ell)}$ denote some mapping of the first $\ell-1$ levels of nodes in W. So, $\tau^{(0)}$ denotes the empty mapping, $\tau^{(1)}$ maps just the root p_0, and $\tau^{(d+1)}$

denotes a mapping of all the vertices. Let us define $L_e(\tau^{(\ell)}, p_{i-1}, p_i)$ as the fractional load on e due to edge (p_{i-1}, p_i) based on the mapping $\tau^{(\ell)}$ thus far. Formally, we define $L_e(\tau^{(\ell)}, p_{i-1}, p_i)$ as follows. If $\ell \geq i+1$, then $\tau^{(\ell)}(p_{i-1})$ and $\tau^{(\ell)}(p_i)$ are defined. Now we set

$$L_e(\tau^{(\ell)}, p_{i-1}, p_i) \;:=\; \frac{\mathcal{F}_e\left(p_0, \tau^{(\ell)}(p_0), \ldots, p_{i-1}, \tau^{(\ell)}(p_{i-1}), p_i, \tau^{(\ell)}(p_i)\right)}{y(p_0, \tau^{(\ell)}(p_0), \ldots, p_i, \tau^{(\ell)}(p_i))} \quad \text{if } \ell \geq i+1.$$

Otherwise (i.e., for $\ell \leq i$),

$$L_e(\tau^{(\ell)}, p_{i-1}, p_i) := \sum_{v_\ell, \ldots, v_i} \frac{\mathcal{F}_e\left(p_0, \tau^{(\ell)}(p_0), \ldots, p_{\ell-1}, \tau^{(\ell)}(p_{\ell-1}), p_\ell, v_\ell, \ldots, p_i, v_i\right)}{y(p_0, \tau^{(\ell)}(p_0), \ldots, p_{\ell-1}, \tau^{(\ell)}(p_{\ell-1}))}.$$

We note that by conditional congestion constraints (15), $L_e(\tau^{(\ell)}, p_{i-1}, p_i)$ is well defined and is always upper bounded by c_e.

A crucial observation is the following.

Lemma 4.1. *Let $\tau^{(\ell)}$ be any arbitrary mapping of vertices in the first $\ell-1$ levels. Let $\tau^{(\ell+1)}$ be obtained from $\tau^{(\ell)}$ by applying our rounding procedure to ℓ-level vertices. Then, for any substrate edge $e \in E$ and workload edge $(p_{i-1}, p_i) \in F$,*

$$\mathbb{E}[L_e(\tau^{(\ell+1)}, p_{i-1}, p_i)] = L_e(\tau^{(\ell)}, p_{i-1}, p_i),$$

where expectation is taken over the randomness in the rounding procedure applied to ℓ-level vertices.

Proof. First, if $\ell > i$, then the mappings of p_{i-1} and p_i are already fixed in $\tau^{(\ell)}$ and the lemma is trivially true, so we assume that $\ell \leq i$.

Let p_ℓ denote the ℓ-level node on the path from p_0 to p_i. By the rounding procedure, the probability that p_ℓ is mapped to v conditioned on the mapping $\tau^{(\ell)}$ until level $\ell - 1$,

$$\Pr\left[\tau^{(\ell+1)}(p_\ell) = v \,|\, \tau^{(\ell)}\right] \;=\; \frac{y(p_0, \tau^{(\ell)}(p_0), \ldots, p_{\ell-1}, \tau^{(\ell)}(p_{\ell-1}), p_\ell, v)}{y(p_0, \tau^{(\ell)}(p_0), \ldots, p_{\ell-1}, \tau^{(\ell)}(p_{\ell-1}))}. \tag{20}$$

Thus,

$$\begin{aligned}\mathbb{E}\left[L_e(\tau^{(\ell+1)}, p_{i-1}, p_i)\right] &= \sum_{v_\ell} \Pr\left[\tau^{(\ell+1)}(p_\ell) = v_\ell \,|\, \tau^{(\ell)}\right] \cdot L_e\left(\tau^{(\ell+1)}, p_{i-1}, p_i\right) \\ &= L_e\left(\tau^{(\ell)}, p_{i-1}, p_i\right),\end{aligned}$$

where the equality in the last step follows by (20) and the definition of L_e (in the regime $\ell \leq i$). ◻

Given a partial mapping $\tau^{(\ell)}$ of nodes on first $\ell - 1$ levels, let $L_e(\tau^{(\ell)}) = \sum_{(p_{i-1},p_i)\in F} L_e(\tau^{(\ell)}, p_{i-1}, p_i)$ denote the total load on edge $e \in E$ due to all edges in F.

Definition 4.1. *A partial mapping $\tau^{(\ell)}$ is said to be* good *if $L_e(\tau^{(\ell)}) \leq (16d(\ell+1)\log nd) \cdot c_e$ for every edge $e \in E$.*

Clearly, the empty mapping $\tau^{(0)}$ is good, since

$$L_e(\tau^{(0)}) \;=\; \sum_{(p_{i-1},p_i)\in F} \;\sum_{v_0,\ldots,v_i} \mathcal{F}_e(p_0, v_0, \ldots, p_i, v_i),$$

which, by the global congestion constraint (13) in the LP, is at most c_e.

LEMMA 4.2. *For any $\ell \in \{0, \dots, d\}$,*

$$\Pr\left[\tau^{(\ell+1)} \textit{ is good} \mid \tau^{(\ell)} \textit{ is good}\right] \;\geq\; 1 - \frac{1}{(dn)^4}.$$

Proof. Let E'' denote the edges of H induced on the vertices of the first $\ell - 1$ levels. For any vertex p_ℓ in level ℓ (with $p_0, \dots, p_{\ell-1}, p_\ell$ being its path from the root), let $E'(p_\ell)$ denote the set of edges in the subtree rooted at p_ℓ plus the edge $(p_{\ell-1}, p_\ell)$. For any subset S of edges, define $L_e(\tau^{(\ell)}, S) = \sum_{(p_{i-1},p_i)\in S} L_e(\tau^{(\ell)}, p_{i-1}, p_i)$; then $L_e(\tau^{(\ell+1)}, S)$ is defined similarly. Since E'' and $\{E'(p_\ell)\}$ partition edges of H,

$$L_e\left(\tau^{(\ell)}\right) \;=\; L_e\left(\tau^{(\ell)}, E''\right) + \sum_{p_\ell} L_e\left(\tau^{(\ell)}, E'(p_\ell)\right).$$

A similar equality holds for $L_e(\tau^{(\ell+1)})$. Recall that $\tau^{(\ell)}$ is a fixed mapping for levels until $\ell - 1$. The randomness is in the choice of the mapping for ℓ-level vertices, which gives $\tau^{(\ell+1)}$. So $L_e(\tau^{(\ell+1)}, E'') = L_e(\tau^{(\ell)}, E'')$ is a deterministic quantity. Next, note that each $L_e(\tau^{(\ell+1)}, E'(p_\ell))$ depends only on the choice $\tau^{(\ell+1)}(p_\ell)$; i.e., for different vertices p_ℓ, the $L_e(\tau^{(\ell+1)}, E'(p_\ell))$'s are *independent* random variables.

Moreover, by Lemma 4.1, the expectation $\mathbb{E}[L_e(\tau^{(\ell+1)}, E'(p_\ell))] = L_e(\tau^{(\ell)}, E'(p_\ell))$ over the random choice of $\tau^{(\ell+1)}(p_\ell)$. Finally, by the conditional congestion LP constraints (15), for *any* choice of $\tau^{(\ell+1)}(p_\ell)$ it holds that $L_e(\tau^{(\ell+1)}, E'(p_\ell)) \leq c_e$.

Thus $L_e(\tau^{(\ell+1)}) - L_e(\tau^{(\ell+1)}, E'') = \sum_{p_\ell} L_e\left(\tau^{(\ell+1)}, E'(p_\ell)\right)$ is the sum of independent $[0, c_e]$ bounded random variables, having mean $\sum_{p_\ell} L_e\left(\tau^{(\ell)}, E'(p_\ell)\right) = L_e\left(\tau^{(\ell)}\right) - L_e\left(\tau^{(\ell)}, E''\right)$, which is at most $16d(\ell+1)\log nd \cdot c_e - L_e\left(\tau^{(\ell)}, E''\right)$ as $\tau^{(\ell)}$ is good. As $L_e(\tau^{(\ell+1)}, E'') = L_e(\tau^{(\ell)}, E'')$ is fixed, using the Chernoff bound (Lemma 1.1 with $\epsilon = 1/(\ell+1)$), it follows that

$$\Pr\left[L_e(\tau^{(\ell+1)}) > 16d(\ell+2)\,\log nd \cdot c_e\right] \leq \exp\left(-16d(\ell+1)\frac{\log nd}{4(\ell+1)^2}\right) \leq \frac{1}{(dn)^4},$$

where the last inequality uses that $\ell + 1 \leq d$. ∎

Applying Lemma 4.2 inductively, it follows that

$$\Pr\left[\tau^{(d+1)} \text{ is good for edge } e\right] \geq 1 - \frac{d+1}{(nd)^4} > 1 - \frac{1}{n^4}; \tag{21}$$

i.e., $L_e\left(\tau^{(d+1)}\right) \leq 32d^2\log(nd)\cdot c_e$ with probability at least $1 - \frac{1}{n^4}$. This completes the analysis of the vertex mapping. We now analyze the edge mapping step.

We condition on the full vertex mapping and denote it by $\tau := \tau^{(d+1)}$. Fix any edge $e \in E$, and observe that the actual load on e after the *edge mapping* is

$$A_e := \sum_{(p_{i-1},p_i)\in F} b(p_{i-1}, p_i)\cdot I[e \in \sigma(p_{i-1}, p_i)];$$

here $\sigma(p_{i-1}, p_i)$ is the random $\tau(p_{i-1}) - \tau(p_i)$ path assigned to edge (p_{i-1}, p_i), and $I[e \in \sigma(p_{i-1}, p_i)]$ is the indicator random variable for the event "$e \in \sigma(p_{i-1}, p_i)$." Recall that the random variables $I[e \in \sigma(p_{i-1}, p_i)]$ are independent, and by (19),

$$\mathbb{E}[A_e] \;=\; \sum_{(p_{i-1},p_i)\in F} \frac{\mathcal{F}_e\left(p_0, \tau(p_0), \dots, p_{i-1}, \tau(p_{i-1}), p_i, \tau(p_i)\right)}{y(p_0, \tau(p_0), \dots, p_i, \tau(p_i))} \;=\; L_e(\tau).$$

Moreover, for each $(p_{i-1}, p_i) \in F$, note that every edge in path $\sigma(p_{i-1}, p_i)$ has capacity at least $b(p_{i-1}, p_i)$: this is by definition of the flow-network $N(p_{i-1}, \tau(p_{i-1}), p_i, \tau(p_i))$ from which the path is sampled.[4] Thus it follows that A_e is the sum of independent $[0, c_e]$ bounded random variables, with mean $L_e(\tau)$. Observe that the mean is at most $32d^2 \log(nd) \cdot c_e$ when *conditioned* on τ being *good for edge* e. So,

$$\begin{aligned}
&\Pr[A_e \geq 64d^2 \log(nd) \cdot c_e] \\
&\quad \leq \Pr\left[A_e \geq 64d^2 \log(nd) \cdot c_e \mid \tau \text{ good for } e\right] + \Pr\left[\tau^{(d+1)} \text{ not good for } e\right] \leq \frac{2}{n^4}.
\end{aligned}$$

The first term is upper bounded using Lemma 1.1 (with $\epsilon = 1$) for the edge mapping, and the second term is from (21). Taking a union bound over the possible n^2 edges implies that the maximum edge congestion is $O(d^2 \cdot \log(nd))$ with probability at least $1 - \frac{1}{n^2}$. An identical analysis works for the node congestion, and we obtain Theorem 4.1 and hence Theorem 1.1.

5. Complete graph workloads.

In this section we consider the GraphMap problem when the workloads are complete graphs with uniform processing and traffic demands and the substrate is a general graph. By scaling edge capacities in G, we can assume (without loss of generality) that the workload H is a complete graph K_r with unit traffic demand between every pair of vertices. Similarly, by scaling node capacities in G, we assume that H has unit node demands.

We first present an algorithm for SingleMap where the substrate is a tree and the workload is a uniform complete graph. Later we show that the Räcke decomposition tree can be used to obtain results on general substrates. If only splittable routing is needed, the Räcke decomposition can be used directly; however, we show that we can also obtain unsplittable routings with some more work. Using our general framework, this gives polylogarithmic ratio offline and online algorithms for GraphMap. However, as Räcke decomposition is an intermediate step, we need to take more care in the reduction to SingleMap.

5.1. Exact algorithm for SingleMap on tree substrate.

Here we consider a special class of substrate graphs, given by a tree $T = (V', E')$. We denote the leaves in T by $V \subseteq V'$ and assume that processes can be mapped only to leaves. This assumption is without loss of generality (we can just introduce a new leaf-node adjacent to each original node of the tree). We consider this type of tree because the Räcke decomposition tree (which we will use later) of any graph $G = (V, E)$ is of this form (recall Definition 1.1).

In the SingleMap problem, there are capacities $c : E' \to \mathbb{R}_+$ on edges and $u : V \to \mathbb{R}_+$ on leaves (nonleaf nodes have zero capacity). In addition, there are cost functions $\alpha : E \to \mathbb{R}_+$ and $\beta : V \to \mathbb{R}_+$. Since the substrate is a tree, a mapping is entirely determined by an assignment of H-vertices to V. The goal is to find such an assignment satisfying node and edge capacities with minimum cost. We show now how this problem can be solved exactly by dynamic programming.

Since the workload is a complete graph with unit demands, the load on any edge $e \in T$ is determined by the number of H-vertices assigned to either side of e in the tree: if the two components in $T \setminus \{e\}$ contain ℓ and $r - \ell$ vertices from H, then the load on e equals $\ell \cdot (r - \ell)$.

[4]This is the main reason that flow variables were restricted to "high capacity" edges; see the LP constraint (7).

Root the tree T at an arbitrary nonleaf vertex $s \in V' \setminus V$. By splitting high-degree vertices (introducing dummy vertices connected by edges of infinite capacity and zero cost), we can assume that each nonleaf vertex in T has at most two children; this assumption is made merely to simplify the description of the dynamic program.

For any $v \in V'$ let T_v denote the subtree of T rooted at vertex v. We now define the entries in the dynamic programming table. For any $v \in V'$ and $0 \le \ell \le r$ let $M[v, \ell]$ denote the least cost partial solution that assigns ℓ nodes of H to the leaves in T_v. The cost $M[v, \ell]$ is the total of node and edge costs over all nodes and edges in T_v; note that the edge costs also involve traffic between nodes of T_v and $V' \setminus T_v$. Clearly, the value $M[s, r]$ at the root s equals the optimum of the SingleMap instance.

We can use the following recurrence to compute $M[v, \ell]$. For all leaves $v \in V$ and $0 \le \ell \le r$, set

$$M[v,\ell] = \begin{cases} \beta_v \cdot \ell & \text{if } \ell \le u_v, \\ \infty & \text{otherwise.} \end{cases}$$

For any nonleaf vertex $v \in V'$ with children v_1 and v_2, and $0 \le \ell \le r$, set

$$M[v,\ell] = \min_{\ell_1,\ell_2} \left\{ M[v_1,\ell_1] + M[v_2,\ell_2] + \alpha_{(v,v_1)} \cdot \ell_1(r-\ell_1) + \alpha_{(v,v_2)} \cdot \ell_2(r-\ell_2) \right\},$$

where the minimum is over all $0 \le \ell_1, \ell_2 \le r$ such that $\ell_1 + \ell_2 = \ell$, $\ell_1(r-\ell_1) \le c_{(v,v_1)}$, and $\ell_2(r-\ell_2) \le c_{(v,v_2)}$. Here ℓ_1, ℓ_2 are the numbers of H-vertices in the subtrees T_{v_1} and T_{v_2}. $M[v,\ell]$ is obtained by enumerating over all (at most r many) possibilities for ℓ_1 and ℓ_2. The constraints on ℓ_1 and ℓ_2 ensure that the loads on edges (v, v_1) and (v, v_2) do not exceed their capacity. If there is no feasible solution $\{\ell_1, \ell_2\}$, then set $M[v,\ell] = \infty$.

This dynamic program can be implemented in $O(nr^2)$ time.

5.2. Offline algorithm for GraphMap. Here the substrate $G = (V, E)$ is general, and each workload is a complete graph with unit demands.

The algorithm guesses value $\Lambda \in [\mathsf{Opt}, 2\mathsf{Opt}]$, where Opt is the optimal value—we can try all (polynomially many) possibilities. Then we apply the procedure of [15] to substrate *G restricted to edges and nodes of capacity at least* $1/\Lambda$ to obtain a Räcke decomposition tree $T(\Lambda)$; recall Definition 1.1. Note that the optimal solution uses only edges/nodes of capacity at least $1/\Lambda$ in G since $\mathsf{Opt} \le \Lambda$. This is why we restrict our attention to edges/nodes of G with sufficiently high capacity.

Now we consider the offline GraphMap instance on substrate $T(\Lambda)$, for which there is an $O(\frac{\log n}{\log\log n})$-approximation algorithm using the SingleMap algorithm above within the offline framework (section 2). By property 2 in Definition 1.1, the optimal value of the GraphMap instance on Räcke tree $T(\Lambda)$ is at most the GraphMap optimal value on G, which is at most Λ. So we obtain a mapping on $T(\Lambda)$ having congestion $O(\frac{\log n}{\log\log n}) \cdot \Lambda$. Using property 3 in Definition 1.1, this yields a splittable-routing solution in G, where the following hold:

- The node congestion is $O(\frac{\log n}{\log\log n}) \cdot \Lambda$.
- The edge congestion is $O(\log^2 n \log\log n) \cdot O(\frac{\log n}{\log\log n}) \cdot \Lambda = O(\log^3 n) \cdot \Lambda$, since $T(\Lambda)$ is an $O(\log^2 n \log\log n)$-approximate Räcke tree.
- Every edge used in this solution has capacity $\ge \frac{1}{\Lambda}$, by definition of $T(\Lambda)$.

Note that, in this solution, each demand edge e is mapped to a unit flow $\mathcal{F}_e$ between its end-points. The total usage of each edge $e' \in G$ is at most $O(\log^3 n)\, \Lambda \cdot c_{e'}$. Finally, each demand edge e sends unit flow along *one path* between its end-points chosen independently according to a flow-path decomposition of $\mathcal{F}_e$. The total load on any edge $e' \in G$ is the sum of independent $\{0,1\}$ random variables, due to the guarantee that $c_{e'} \geq 1/\Lambda$ falls also into the range $[0, \Lambda c_{e'}]$. The mean load on edge e' is $O(\log^3 n)\,\Lambda \cdot c_{e'}$. Thus, by Lemma 1.1 (with, say, $\epsilon = 1$), it follows that the final congestion is $O(\log^3 n) \cdot \Lambda$ with high probability. This proves the first part of Theorem 1.2.

5.3. Online algorithm for GraphMap. Recall that the substrate is a general graph $G = (V, E)$, and each workload is a complete graph with unit demands. Here workloads arrive (with known durations) and depart over time. We assume that all times are integral, and we let R denote the time horizon. We present a polylogarithmic competitive *randomized* online algorithm. We obtain an online algorithm for GraphMap on tree substrates using the SingleMap algorithm above and the framework in section 3. This immediately translates to an online algorithm for *splittable* routing on general substrates: we just run the online algorithm on the Räcke tree and use the flow templates (property 3 of Definition 1.1) to route the traffic-demands in G. To obtain the algorithm for unsplittable routing, we first give a stronger online algorithm for splittable routing that has (roughly) this additional *support* property: all flow-paths of a workload use edges of capacity at least $\frac{1}{\mathsf{Opt}}$, where Opt is the optimal offline value of the current input sequence. Then we obtain the unsplittable routing by simple randomized rounding of the splittable routing (as in the offline algorithm of the previous subsection); this is why the final online algorithm is randomized.

Stronger online algorithm for splittable routing. We first describe an algorithm that also assumes an upper bound Λ on the optimal (offline) value of the input sequence; later we show how to remove this assumption. Again let $T(\Lambda)$ denote the Räcke decomposition tree from the procedure of [15] applied to substrate G restricted to nodes and edges of capacity at least $1/\Lambda$. As in the offline algorithm, it follows that the optimal value of the input sequence on substrate $T(\Lambda)$ is at most Λ, since the optimal value on substrate G is at most Λ (by our assumption).

Using the SingleMap algorithm within the framework in section 3 gives an online algorithm $\mathcal{A}(\Lambda)$ on $T(\Lambda)$ that, assuming the optimal offline value on $T(\Lambda)$ is at most Λ, produces a solution of congestion at most $\gamma \cdot \Lambda$, where $\gamma = O(\log(nD))$. Again, using the Räcke flow templates (property 3 of Definition 1.1), the online mapping on $T(\Lambda)$ yields an online splittable-routing solution in G, where the following hold:

- The node congestion (at any time) is $O(\log(nD)) \cdot \Lambda$.
- The edge congestion (at any time) is $O(\log(nR)\,\log^2 n \log\log n) \cdot \Lambda$, since $D \leq R$.
- *Support property:* Every edge used in this solution has capacity $\geq \frac{1}{\Lambda}$, by definition of $T(\Lambda)$.

Removing assumption of knowing $\boldsymbol{\Lambda}$. We use the standard doubling approach [9] with one extra step (to handle the support property above). We partition the execution of the algorithm into phases determined by the value of Λ. Initially Λ is set to some lower bound $\lambda_0 > 0$ on the optimal value of the first workload. The value of Λ will always be of the form $\lambda_i := 2^i \cdot \lambda_0$ for some integer $i \geq 0$. When $\Lambda = \lambda_i$, we say that the algorithm is in phase i. For each $i \geq 0$, we construct an $O(\log^2 n \log\log n)$-approximate Räcke tree $T(\lambda_i)$ using the algorithm of [15]; these trees are all disjoint.

Suppose that the algorithm is in phase i and receives a new workload H. We attempt to map H into tree $T(\lambda_i)$ using algorithm $\mathcal{A}(\lambda_i)$ from above. If the new congestion on $T(\lambda_i)$ remains at most $\gamma \cdot \lambda_i$, then we accept this mapping for H and continue in phase i. Else (i.e., new congestion $> \gamma \cdot \lambda_i$), we reject this map for H, double Λ, and enter phase $i+1$ with H as the new workload.

Let ℓ denote the last phase. Let σ_i denote the input sequence received in any phase $0 \le i \le \ell$. Observe that the optimal value of $\sigma_{\ell-1}$ on $T(\lambda_{\ell-1})$ (and hence G) is at least $\lambda_{\ell-1}$; otherwise the algorithm would not have progressed to phase ℓ. So the offline optimal value of the entire input is at least $\lambda_\ell/2$. We now bound the algorithm's congestion induced over all phases. Note that the algorithm's congestion due to σ_i on $T(\lambda_i)$ is at most $\gamma \cdot \lambda_i \; \forall\, 0 \le i \le \ell$. By property 3 of Definition 1.1 the edge-congestion due to σ_i on G is at most $\gamma \cdot O(\log^2 n \log\log n) \cdot \lambda_i$. (The node-congestion due to σ_i on G remains at most $\gamma \cdot \lambda_i$.) Thus the maximum congestion on G is $\gamma \cdot O(\log^2 n \log\log n) \cdot \lambda_\ell$.

Randomized rounding. Finally, we obtain an online algorithm for unsplittable routing by randomly rounding the above splittable routing (which has the support property). We perform the rounding for all edges of a workload immediately upon its arrival. By the support property, in any phase i the algorithm uses only edges with capacity at least $1/\lambda_i$. Thus if ℓ denotes the last phase, every edge used in the splittable routing has capacity at least $1/\lambda_\ell$ (as λ_i's are increasing). Also for any edge $e' \in G$, as shown above, its total usage in the splittable routing is $O(\log^2 n \log\log n \log(nR))\, \lambda_\ell \cdot c_{e'}$. So the total load on $e' \in G$ at any point of time is the sum of independent $\{0,1\}$ random variables with mean $O(\log^2 n \log\log n \log(nR))\, \lambda_\ell \cdot c_{e'}$. As $\lambda_\ell \cdot c_{e'} \ge 1$ for each edge with nonzero load, applying Lemma 1.1 (with $\epsilon = 1$), it follows that for any $e' \in G$ and time t,

$$\Pr\left[\text{congestion of } e' \text{ at time } t \text{ is greater than } O(\log^2 n \log\log n \log(nR)) \cdot \lambda_\ell\right] \le \frac{1}{(n+R)^4}.$$

Taking a union bound over all these events (at most $n^2 \cdot R$ in number), the congestion of the online mapping is $O(\log^2 n \log\log n \log(nR)) \cdot \mathsf{Opt}$ with high probability, where Opt is the optimal value. This completes the proof of Theorem 1.2.

6. Concluding remarks.

We have given a general framework for solving a natural graph mapping problem arising in cloud computing. We then applied this framework to obtain offline and online approximation algorithms for workloads given by depth-d trees and complete graphs. In the paper, for notational simplicity we focused on the case in which each server has a single resource. We note here that most of our algorithms easily extend to the setting of $r > 1$ resources (e.g., CPU, memory, disk, etc.), i.e., when there is an r-dimensional capacity constraint at each vertex. The modifications are as follows:

- The definition of SingleMap now has demand, capacity, and cost for each $\langle$vertex, resource$\rangle$ pair.
- In the algorithm for offline GraphMap (section 2) the configuration LP also has capacity constraints for each $\langle$vertex, resource$\rangle$ pair; the approximation ratio becomes $O\big((\rho_1+\rho_2) \cdot \frac{\log(nr)}{\log\log(nr)}\big)$ since we use Lemma 1.1 and union over $O(nr + n^2)$ events.
- In the online algorithm (section 3), we work with groundset U consisting of all edges and $\langle$vertex, resource$\rangle$ pairs; so $|U| = O(nr + n^2)$. Consequently the competitive ratio becomes $O\left((\rho_1 + \rho_2) \cdot \log(nr)\right)$.
- For SingleMap on depth-d tree workloads (section 4), the LP congestion constraints also handle $\langle$vertex, resource$\rangle$ pairs. Again the guarantee becomes

$O\left(d^2 \cdot \log(ndr)\right)$ due to randomized rounding.

- For complete-graph workloads, where all nodes have an identical r-dimensional resource vector and every pair of nodes has uniform traffic demand, the GraphMap problem reduces easily to the $r = 1$ case. Thus our algorithms from section 5 apply.

Hardness results. Interestingly, even though the GraphMap problem is closely related to many other hard problems, we are unable to show any nontrivial hardness results for GraphMap itself. Roughly speaking, the ability to map nodes arbitrarily in GraphMap seems to lead to too much flexibility in finding good solutions, which makes the reductions fail.

The best hardness of approximation for GraphMap that we can show is 1.8 (which can most likely be pushed to $2 - \epsilon$) via the *balanced separator* problem. In particular, we use Corollary IV.6 in [24], which implies (assuming the small-set expansion (SSE) conjecture) that it is NP-hard to distinguish the following two cases for a given graph $G = (V, E)$ and value d:

- Yes case: there is a subset $S \subseteq V$ with $|S| = n/2$ and $\delta_G(S) \leq d$.
- No case: every subset $S \subseteq V$ with $0.1n \leq |S| \leq 0.9n$ has $\delta_G(S) \geq 2d$.

By considering the GraphMap problem with workload G and a single-edge substrate (with node capacities $n/2$ and edge capacity d), we find that GraphMap is SSE-hard to approximate to better than factor 1.8. Notice that this reduction from balanced-separator to GraphMap (where the substrate graph is a single edge) cannot give any hardness of approximation better than factor 2, as a trivial 2-approximation algorithm in this case is to map all vertices to a single node of the substrate.

The best lower bound for online GraphMap that we can show is 2.438 using *online load balancing with related machines*, for which [8] showed that any deterministic online algorithm is at least 2.438 competitive. Note that this is an instance of online GraphMap where the substrate consists of only isolated nodes and the workloads are singleton vertices.

Acknowledgment. We thank the anonymous reviewers for comments that helped improve the presentation of the paper.

REFERENCES

[1] W. P. Adams and T. A. Johnson, *Improved linear programming-based lower bounds for the quadratic assignment problem*, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 16 (1994), pp. 43–77.
[2] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley, New York, 2008.
[3] S. Arora, S. Rao, and U. V. Vazirani, *Expander flows, geometric embeddings and graph partitioning*, J. ACM, 56 (2009), 5.
[4] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts, *On-line routing of virtual circuits with applications to load balancing and machine scheduling*, J. ACM, 44 (1997), pp. 486–504.
[5] Y. Azar, B. Kalyanasundaram, S. A. Plotkin, K. Pruhs, and O. Waarts, *Online load balancing of temporary tasks*, in Proceedings of the Third Workshop on Algorithms and Data Structures (WADS), Montreal, 1993, pp. 119–130.
[6] N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. Naor, and R. Schwartz, *Min-max graph partitioning and small set expansion*, in Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), Palm Springs, 2011, pp. 17–26.
[7] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer, *Minimum congestion mapping in a cloud*, in Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), San Jose, 2011, pp. 267–276.

[8] P. Berman, M. Charikar, and M. Karpinski, *On-line load balancing for related machines*, J. Algorithms, 35 (2000), pp. 108–121.

[9] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[10] E. Cela, *The Quadratic Assignment Problem: Theory and Algorithms*, Springer, New York, 1998.

[11] N. Chowdhury, M. Rahman, and R. Boutaba, *Virtual network embedding with coordinated node and link mapping*, in Proceedings of the 28th IEEE Conference on Computer Communications (INFOCOM), 2009, pp. 783–791.

[12] J. Chuzhoy and S. Li, *A polylogarithmic approximation algorithm for edge-disjoint paths with congestion* 2, in Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), New Brunswick, NJ, 2012, pp. 233–242.

[13] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, John Wiley and Sons, New York, 1998.

[14] D. Golovin, A. Gupta, B. M. Maggs, F. Oprea, and M. K. Reiter, *Quorum placement in networks: Minimizing network congestion*, in Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC), Denver, 2006, pp. 16–25.

[15] C. Harrelson, K. Hildrum, and S. Rao, *A polynomial-time tree decomposition to minimize congestion*, in Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), San Diego, 2003, pp. 34–43.

[16] R. Hassin, A. Levin, and M. Sviridenko, *Approximating the minimum quadratic assignment problems*, ACM Trans. Algorithms, 6 (2009), 18.

[17] F. Thomson Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.

[18] K. Makarychev, R. Manokaran, and M. Sviridenko, *Maximum quadratic assignment problem: Reduction from maximum label cover and lp-based approximation algorithm*, in Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP), Bordeaux, 2010, pp. 594–604.

[19] X. Meng, V. Pappas, and L. Zhang, *Improving the scalability of data center network with traffic-aware virtual machine placement*, in Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM), San Diego, 2010, pp. 1154–1162.

[20] V. Nagarajan and M. Sviridenko, *On the maximum quadratic assignment problem*, Math. Oper. Res., 34 (2009), pp. 859–868.

[21] H. Räcke, *Minimizing congestion in general networks*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), Vancouver, 2002, pp. 43–52.

[22] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.

[23] P. Raghavendra and D. Steurer, *Graph expansion and the unique games conjecture*, in Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC), Cambridge, MA, 2010, pp. 755–764.

[24] P. Raghavendra, D. Steurer, and M. Tulsiani, *Reductions between expansion problems*, in Proceedings of the 27th Conference on Computational Complexity (CCC), 2012, pp. 64–73.

[25] H. D. Sherali and W. P. Adams, *A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems*, SIAM J. Discrete Math., 3 (1990), pp. 411–430.

[26] M. Yu, Y. Yi, J. Rexford, and M. Chiang, *Rethinking virtual network embedding: Substrate support for path splitting and migration*, ACM SIGCOMM Computer Commun. Rev., 38 (2008), pp. 19–29.

[27] Y. Zhu and M. Ammar, *Algorithms for assigning substrate network resources to virtual network components*, in Proceedings of the 25th IEEE Conference on Computer Communications (INFOCOM), Barcelona, 2006, pp. 1–12.