Minimum Congestion Mapping in a Cloud*

Nikhil Bansal† Kang-Won Lee† Viswanath Nagarajan† 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 denoting 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 to solve the general problem. In particular, an α-approximation 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 class 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 based on a strong LP relaxation inspired by the *Sherali-Adams* hierarchy. For complete graphs, we give a poly-logarithmic approximation algorithm using Räcke decompositions.

1 Introduction

In cloud computing, the underlying resource is a physical network (also called the *substrate*) consisting of servers that are inter-connected 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.

The allocation 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 to servers, and (b) *path-mapping*, the assignment of each communication request (i.e. edge between two processes) 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 like that the mapping should not cause the load on any node or edge to exceed it capacity. However if this constraint is enforced strictly, the

*A preliminary version of this paper appeared as [6]. Research was sponsored in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) 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 is also supported through participation in the Measurement Science for Cloud Computing sponsored by the National Institute of Standards and Technology (NIST) under Agreement Number 60NANB10D003.

†IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.

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

problem can be shown hard to approximate to within any reasonable factor. This holds even for simple workloads such as stars 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 node or edge is defined as the ratio of its load to 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, 13, 20] 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 $n/2$. This reduces to partitioning the vertices of the workload H into near-balanced parts such that the resulting cut is minimized. This is an extensively studied graph partitioning problem, and the best known approximation ratio is $O(\sqrt{\log n})$ [2].

- *Cut-width:* The substrate is a line on n vertices with equal capacity edges. This reduces to finding an ordering $v_1, \ldots, v_n$ of the vertices in the workload H such that $\max_{i=1}^{n} \delta_H(v_1, \ldots, v_i)$ is minimized. The best known approximation ratio is $O(\log^{3/2} n)$ [18].

- *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, a basic problem in distributed computing. This problem has been studied only recently [24, 5] and an $O(\sqrt{\log n \log k})$-approximation algorithm is known [5].

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, that we call SingleMap, suffices to solve both the offline and online GraphMap problem. In addition to this connection, clearly 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 corresponding 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 with each other. We require that the processing requirements be identical and also the communication requirement be identical for every pair of processes[2].

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

1.1 Model

The General Mapping Problem: The substrate is a 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 a 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:

- $\pi : W \to V$ 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 alternate model 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; and congestion of v is N_v/u_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 the 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 eg. [4]), 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 poly-logarithmic approximation factor) that the substrate G is always a tree [22, 16]. However this reduction to trees is not applicable in the unsplittable routing model that we consider. Nevertheless, we make use of Räcke decomposition trees [22, 16] in our algorithm for complete-graph workloads.

Single Workload Mapping: This is an important subroutine that is useful in obtaining algorithms for both the offline and online mapping problems. The input to SingleMap consists of the following:

- A 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). The mapping is called *valid* if it respects all edge and node capacities, i.e. $L_e \leq c_e \ \forall e \in E$ and $N_v \leq u_v \ \forall v \in V$. The goal is to find a valid mapping of H into G that minimizes the total cost $\sum_{e \in E} \alpha_e \cdot L_e + \sum_{v \in V} \beta_v \cdot N_v$.

An algorithm for SingleMap is said to be a (ρ_1, ρ_2) *bicriteria approximation* algorithm if: (1) it produces a solution of cost at most ρ_1 times the optimum, and (2) such that all edge and node capacities are satisfied within a ρ_2 factor.

We note that the objective in SingleMap is not congestion (as in GraphMap), but the cost of the mapping.

1.2 Our Results and Techniques

First, we describe the general frameworks for designing both offline and online algorithms of GraphMap, assuming a (ρ_1, ρ_2) bicriteria approximation algorithm for SingleMap. In particular,

- For offline GraphMap (where all workloads are given upfront) 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 [3]. We discuss the differences from [3] in the next Subsection 1.3.

These results appear in Section 2 and Section 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 (trivially) solved optimally by 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* $\left(2, O(d^2 \cdot \log nd)\right)$ *bicriteria approximation algorithm for* SingleMap *on* d*-depth tree workloads, that 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 quasipolynomial, poly-logarithmic 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 [13, 20] 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 [22, 16]. However some more care is required in this reduction since the Räcke tree only provides a splittable routing, and we finally want an unsplittable routing.

Theorem 1.2 *For* GraphMap *under uniform complete-graph workloads there is:*

- *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 [17] or maximized [21, 19]. On the other hand, the virtual

circuit routing problem [3] deals with the second issue (although only for single edge workloads). Here, the mapping of the endpoints of the workload edge are given, and the goal is to map the edge to a path in the substrate graph to minimize edge congestion.

Another related problem is minimizing congestion for quorum placement on networks [15], for which a poly-logarithmic approximation is known. This also involves mapping nodes and paths simultaneously (here routing is splittable). However all paths are between one vertex that is fixed (called client) and another (called quorum) that is mapped. In contrast, both end points of a path in SingleMap are mapped vertices. Moreover, the demand between clients and quorums has a "product multicommodity" structure, whereas demands in SingleMap are arbitrary.

The online framework we present for GraphMap uses ideas from the online virtual circuit routing algorithm [3]. In [3] costs on edges are maintained using multiplicative updates and each request is routed along a shortest-path from its source to 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 [3].

A natural approach to mapping nodes in SingleMap is to consider an LP relaxation similar to ones used for quadratic assignment [1, 19]. 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 poly-logarithmic approximation for this problem. This adds to a small list of problems for which lift-and-project LP hierarchies have proved useful in obtaining algorithms. Some other examples are graph coloring [11], independent set in 3-uniform hypergraphs [12], dense-k-subgraph [8],and max-min degree arborescence [7].

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 $O((\rho_1+\rho_2)\cdot\frac{\log n}{\log\log n})$ 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 $H_1, \ldots, H_k$ 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 $i \in [k]$ 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.

$$\begin{aligned} &\text{minimize} \quad 0 \\ &\text{subject to} \sum_{\tau\in\mathcal{F}_i} x_i(\tau) \geq 1 \qquad \forall i \in [k] \\ &\sum_{i=1}^{k}\sum_{\tau\in\mathcal{F}_i} \ell(e,\tau)\cdot x_i(\tau) \leq c_e \qquad \forall e \in E \end{aligned}$$

$$\sum_{i=1}^{k} \sum_{\tau \in \mathcal{F}_i} \ell(v,\tau) \cdot x_i(\tau) \leq u_v \qquad \forall v \in V$$
$$x_i(\tau) \geq 0 \qquad \forall \tau \in \mathcal{F}_i,\ \forall i \in [k].$$

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 we consider its dual:

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

Observe that given 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 $(\rho_1,\ \rho_2)$ 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:

- 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 $e \in E$, $\sum_{i=1}^{k} \sum_{\tau \in \mathcal{F}_i} \ell(e,\tau) \cdot y_i(\tau) \leq \rho_1 \cdot c_e$.
- For all $v \in V$, $\sum_{i=1}^{k} \sum_{\tau \in \mathcal{F}_i} \ell(v,\tau) \cdot y_i(\tau) \leq \rho_1 \cdot u_v$.
- 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 standard probabilistic tail bounds (as in [23]), 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 $O((\rho_2 + \rho_1)\log(nD))$-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.*

Using standard arguments the term D above can be replaced with the ratio of maximum to minimum durations.

The algorithm is similar to the online algorithm for virtual circuit routing [3, 4, 9]. The idea is that at each time, the algorithm maintains a cost on the edges that is an exponentially increasing function of their load. Upon the arrival of a workload, the solution of an SingleMap instance with these costs determines where this workload will be placed. Since the highly loaded edges are severely penalized, the SingleMap solution will prefer edges with low load.

Notation: Let $H_1, H_2, \ldots, H_k$ 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 s_is are non-decreasing. We assume that all times and durations are integral and $\max_i t_i \leq D$. Also, given 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 $e \in U$. For each workload H_i, let $\mathcal{F}_i$ denote the set of all possible mappings of H_i into G. 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.

$$\max_{e \in U} \quad \max_{h \geq 0} \quad \frac{\sum_{i: s_i \leq h \leq s_i + t_i} \tau_i(e)}{c_e}$$

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

The SingleMap problem can be restated in the above notation: given workload H_i, *costs* $\alpha : U \to \mathbb{R}_+$ and capacities $\overline{c} : U \to \mathbb{R}_+$, find a feasible map $\tau \in \mathcal{F}_i$ minimizing $\sum_{e \in U} \alpha_e \cdot \tau(e)$ such that $\tau(e) \leq \overline{c}_e$ for all $e \in U$. 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 eg. Theorem 12.5 [9]), where the online maintains an upper bound Λ on the optimal congestion thus far. We denote this optimal *offline* solution by maps $\tau_i^* \in \mathcal{F}_i$ for each $i \in [k]$.

Let $\gamma \in (0, 1)$ be a constant to be fixed later. Also let $B := \rho_2$. For any $i \geq 1$, let $\ell_i(e, h)$ denote the load of element e at time h, induced by requests $H_1, \ldots, H_{i-1}$. 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 $s_j \leq h \leq s_j + t_j$.

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

1. Set costs $\alpha_e := \frac{\gamma}{B\, c_e} \cdot \sum_{h=s_i}^{s_i+t_i} \exp\left(\frac{\gamma \ell_i(e,h)}{B \cdot c_e}\right)$ for all $e \in U$.
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 $\ell_{i+1}(e, h) \leftarrow \ell_i(e, h) + \tau_i(e)$ for all $e \in U$ and $s_i \leq h \leq s_i + t_i$.

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 $\tau \in \mathcal{F}_i$ with $\tau(e) \leq B \cdot c_e$ for all $e \in U$, we have:*

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

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 $\tau(e) \in [0,\, B \cdot c_e]$, setting $x = \tau(e)/(Bc_e)$ above,

$$\exp\left(\frac{\gamma \cdot \tau(e)}{Bc_e}\right) - 1 \in \left[\frac{\gamma \cdot \tau(e)}{Bc_e}, 2\frac{\gamma \cdot \tau(e)}{Bc_e}\right].$$

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

For each $e \in U$ and time $h \geq 0$, let $\overline{\ell}(e,h) = \max_i \ell_i(e,h)$ denote the observed load on element e at time h. Observe that for any index i with $s_i > h$, we have $\ell_i(e,h) = \overline{\ell}(e,h)$. Clearly, the objective value of the online algorithm is $\max_{e \in U} \max_{h \geq 0} \overline{\ell}(e,h)/c_e$, that we wish to bound. To this end, for any integer $j \geq 1$ define:

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

We will show that,

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

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

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

by the definition of γ.

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

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

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

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 $\ell_r(e,h) = 0$ for all $e \in U$ and $h \geq (j-1)D$. Thus,

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

■

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*

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

Proof: By definition,

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

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

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

The first equality is by definition of $\tau_i(e,h)$ and the second is by the definition of α_es. Now, recall the algorithm for mapping workload H_i that solves SingleMap instance $\langle H_i, \alpha, c\rangle$. 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:

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

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

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

Combined with (1) and (2), we obtain

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

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:

$$\begin{aligned}&\le\; 2\rho_1\gamma \cdot \sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_{t+1}(e,h)}{B\, c_e}\right) \cdot \left(\sum_{i\in R} \frac{\tau_i^*(e,h)}{B\, c_e}\right) \\ &\le\; \frac{2\rho_1\gamma}{\rho_2} \cdot \sum_{e\in U} \sum_{h=(j-2)D}^{(j+1)D} \exp\left(\frac{\gamma \cdot \ell_{t+1}(e,h)}{B\, c_e}\right).\end{aligned}$$

The second inequality uses $B = \rho_2$ and our assumption that the optimum congestion is at most 1, i.e. $\sum_i \tau_i^*(e,h) \le c_e$.

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

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

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

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

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

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

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

4 Single Workload Mapping on d-Depth Tree Workloads

In this section we prove Theorem 1.1. As mentioned previously, our result in based on an LP formulation inspired by the Sherali-Adams Hierarchy. It is instructive to see why simpler approaches do not seem to work. Clearly, an LP formulation based on assignment variables $x_{p,v}$ which indicate that node p in 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 x_{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, unless strengthened by additional Sherali-Adams type constraints.

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 a feasibility question.

Clearly, any integral mapping must violate the capacity of some edge by a factor of n. However, it turns out there is a feasible solution for Quadratic Assignment LPs [1], that satisfies all the capacities. We set,

$$y(r,v) = \begin{cases} \frac{1}{n} & \text{if } v \in \{a_i\}_{i=1}^n \\ 0 & \text{otherwise} \end{cases}$$

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 that 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 to trees with depth two or more.

4.1 Uniform Star Workload

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 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 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 [14]. 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 non-leaf vertices, since we can again enumerate over all non-leafs 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 (there may be super-logarithmic number of non-leaf vertices).

4.2 Depth d-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. We use $[d] := \{0, 1, \ldots, d\}$. For any $i \in [d]$, 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 a 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 any $i \in [d]$, we use $(p_0, \ldots, p_i)$ to denote a path in H from the root p_0 that contains exactly one vertex in each level $0, 1, \ldots, i$.

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

Assignment Variables: For every index $i \in [d]$, and path $(p_0, p_1, \ldots, p_i)$ in H, and vertices $v_0, \ldots, v_i \in V$, we introduce a variable $y(p_0, v_0, \ldots, p_i, v_i) \in \{0, 1\}$ which we relax to take values in the range $[0, 1]$. In the integral solution, this variable is intended to be 1, if each p_j in the path is mapped to v_j for each $j \in \{0, \ldots, i\}$, and is 0 otherwise. 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 only allow variables where each p is mapped to v such that $g(p) \leq u_v$ (we set the y variable to 0 otherwise).

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 $j \in \{0, \ldots, i\}$. We will denote the variables in this flow instance by $\mathcal{F}_e(p_0, v_0, \ldots, p_i, v_i)$

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 $b(p_{i-1}, p_i)$, the source-vertex is v_{i-1} and sink is v_i. There are flow-variables $\mathcal{F}_e(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i)$ for each edge $e \in G$. The flow on edges $G \setminus N(p_{i-1}, v_{i-1}, p_i, v_i)$ are *fixed to zero*; i.e. only edges $N(p_{i-1}, v_{i-1}, p_i, v_i)$ participate in this flow. The variables satisfy flow-conservation constraints and send

$$y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i) \cdot b(p_{i-1}, p_i) \tag{3}$$

units of flow from v_{i-1} to v_i. One can view $\left\{ \frac{\mathcal{F}_e(p_0, v_0, \ldots, p_i, v_i)}{y(p_0, v_0, \ldots, p_i, v_i)} \right\}_e$ as defining $b(p_{i-1}, p_i)$ units of flow conditioned upon p_j being mapped to v_j for each $j \in \{0, \ldots, i\}$. We note that the network $N(p_{i-1}, v_{i-1}, p_i, v_i)$ itself is independent of where $p_0, \ldots, p_{i-2}$ are mapped.

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 $v_0, \ldots, v_{i-1} \in V$,

$$\sum_{v_i \in V} y(p_0, v_0, \ldots, p_i, v_i) = y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}). \tag{4}$$

   This can be viewed as saying that

$$\frac{y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i)}{y(p_0, v_0, \ldots, p_{i-1}, v_{i-1})}$$

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

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

$$\sum_{v_0 \in V} y(p_0, v_0) = 1. \tag{5}$$

Together (4) and (5) 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 any each edge or vertex in G is at most its capacity.

For each edge $e \in E$, we have

$$\sum_{(p_{i-1}, p_i) \in F} \sum_{v_0, \ldots, v_i} \mathcal{F}_e(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i) \quad \leq \quad c_e. \tag{6}$$

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

$$\sum g(p_i) \cdot y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v) \quad \leq \quad u_v, \tag{7}$$

where the summation is over all indices $i \geq 0$, and paths $(p_0, \ldots, p_i) \in H$ and vertices $v_0, \ldots, v_{i-1} \in V$.

Conditional Congestion Constraints: These final types of constraints are perhaps the least natural, but these are critical to strengthening the LP.

For each index $i \geq 0$, and each path $(p_0, \ldots, p_i)$ and each possible choice of vertices $v_0, \ldots, v_i \in V$, and edge $e \in E$, we add the constraint:

$$\sum_{j \geq i} \sum_{p_{i+1}, v_{i+1}, \ldots, p_j, v_j} \mathcal{F}_e(p_0, v_0, \ldots, p_i, v_i, \ldots p_j, v_j) \quad \leq \quad c_e \cdot y(p_0, v_0, \ldots, p_i, v_i). \tag{8}$$

This constraint is similar to the global edge congestion constraint, except that we condition on event that $p_0, \ldots, p_i$ are mapped to $v_0, \ldots, v_i$ respectively. That is, conditional on $p_0, \ldots, p_i$ being mapped to $v_0, \ldots, v_i$, the total load on e due to mapping edges in subtree rooted at p_i must be no more than c_e. Note that if $y(p_0, v_0, \ldots, p_i, v_i) \in \{0, 1\}$, then this is a valid constraint, and hence the above relaxation is valid.

Similarly, for each vertex $v \in V$, index $i \geq 0$, each path $(p_0, \ldots, p_i)$ and vertices $v_0, \ldots, v_i \in V$, we add:

$$\sum_{j \geq i} \sum_{p_{i+1}, v_{i+1}, \ldots, p_j} g(p_j) \cdot y(p_0, v_0, \ldots, p_i, v_i, \ldots p_j, v) \quad \leq \quad u_v \cdot y(p_0, v_0, \ldots, p_i, v_i). \tag{9}$$

That is, conditional on $p_0, \ldots, p_i$ being mapped to $v_0, \ldots, v_i$, the load on v due to nodes in subtree rooted at p_i must be no more than u_v.

Objective: The objective is to minimize:

$$\sum_{e \in E} \alpha_e \cdot \sum \mathcal{F}_e(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i) + \sum_{v \in V} \beta_v \cdot \sum g(p_i) \cdot y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v). \tag{10}$$

Here, the first summation (over edges) is over all indices $i \geq 1$, all paths $(p_0, \ldots, p_i)$ in H and all vertices $v_0, \ldots, v_i \in V$, and the second summation (for vertices) is over all indices $i \geq 0$, all paths $(p_0, \ldots, p_i)$ and all vertices $v_0, \ldots, v_{i-1} \in V$.

This completes the description of the linear program. 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, as argued above, this LP is a valid relaxation of the SingleMap problem with d-depth tree workloads.

4.3 The Rounding Algorithm

We round the optimal LP solution in d phases, where in the i^{th} 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 $i \in \{1, \ldots, d\}$ do:

   For each vertex p_i at level-i:

   - Let $(p_0, \ldots, p_{i-1}, p_i)$ denote the path from the root to p_i.
   - Set $\tau(p_i) \leftarrow v$ independently with probability:

$$\frac{y(p_0, \tau(p_0), \ldots, p_{i-1}, \tau(p_{i-1}), p_i, v)}{y(p_0, \tau(p_0), \ldots, p_{i-1}, \tau(p_{i-1}))} \tag{11}$$

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, (11) defines a valid (conditional) probability distribution for mapping p_i, due to LP constraint (4).

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

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

  By (3) this gives a probability distribution on $\tau(p_{i-1})$ to $\tau(p_i)$ paths.

- 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 $\sigma(p_{i-1}, p_i)$.

Two simple properties: This completes the description of the rounding procedure. We note here two useful properties of this procedure.

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

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

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

$$\Pr[e \in \sigma(p_{i-1}, p_i) \,|\, \tau(p_0) = v_0, \ldots, \tau(p_i) = v_i] \;=\; \frac{\mathcal{F}_e(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i)}{b(p_{i-1}, p_i) \cdot y(p_0, v_0, \ldots, p_{i-1}, v_{i-1}, p_i, v_i)} \tag{12}$$

4.4 The Analysis

We need to show two things. First, the cost of the mapping is close to optimum. Second, the edge and node congestions are not too high.

Claim 4.1 *The expected cost of the algorithm's mapping $\langle \tau, \sigma \rangle$ equals the optimal LP objective.*

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

Proof:(Claim 4.1) The cost of any mapping $\langle \tau, \sigma \rangle$ is

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

given by the total of node costs and edge costs.

For any level i 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,

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

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

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

which is exactly the second term in the LP objective (10).

We now compute the expected edge-cost. Consider any edge $(p_{i-1}, p_i) \in F$. By (12), and unconditioning over the events $\tau(p_0) = v_0, \ldots, \tau(p_{i-1}) = v_{i-1}, \tau(p_i) = v_i$,

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

So the expected edge-cost is:

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

which is exactly the first term in the LP objective (10). 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 $\ell \in [d]$, let $\tau^{(\ell)}$ denote some mapping of the first $\ell - 1$ levels of nodes in W. So, $\tau^{(0)}$ denotes the empty mapping and $\tau^{(d+1)}$ denote a mapping of all the vertices. Let us define, $L_e(\tau^{(\ell)}, p_{i-1}, p_i)$ as the 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

$$\frac{\mathcal{F}_e\left(p_0, \tau^{(\ell)}(p_0), \ldots, p_{i-1}, \tau^{(\ell)}(p_{i-1}), p_i, \tau^{(\ell)}(p_i)\right)}{y(p_0, \tau^{(\ell)}(p_0), \ldots, p_i, \tau^{(\ell)}(p_i))},$$

and otherwise (i.e. $\ell \leq i$),

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

We note that by conditional congestion constraints (8), $L_e(\tau^{(\ell)}, p_{i-1}, p_i)$ is well-defined and always 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 level ℓ 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 level ℓ vertices.

Proof: Firstly, if $\ell > i$, then the mapping 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 level-ℓ 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 $\ell - 1$,

$$\Pr\left[\tau^{(\ell+1)}(p_\ell) = v \,|\, \tau^{(\ell)}\right] \quad = \quad \frac{y(p_0, \tau^{(\ell)}(0), \ldots, p_{\ell-1}, \tau^{(\ell)}(p_{\ell-1}), p_\ell, v)}{y(p_0, \tau^{(\ell)}(0), \ldots, p_{\ell-1}, \tau^{(\ell)}(p_{\ell-1}))} \tag{13}$$

Thus,

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

where the equality in the last step follows by (13) 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 total load on edge $e \in E$ due to all edges in F. Call $\tau^{(\ell)}$ *good* if $L_e \leq 16d(\ell+1)c_e \log nd$. Clearly, the empty mapping $\tau^{(0)}$ is good, since

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

which by the global congestion constraint in the LP (6) is at most c_e.

Lemma 4.2 *For any $\ell \in [d]$,*

$$\Pr\left[\tau^{(\ell+1)} \textit{ is good} \,|\, \tau^{(\ell)} \textit{ is good}\right] \quad \geq \quad 1 - 1/(dn)^4.$$

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 $p_0, \ldots, p_{\ell-1}, p_\ell$ 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 $L_e(\tau^{(\ell)}, S) = \sum_{(p_{i-1},p_i)\in S} L_e(\tau^{(\ell)}, p_{i-1}, p_i)$; and $L_e(\tau^{(\ell+1)}, S)$ is defined similarly. Since E'' and $\{E'(p_\ell)\}$ partition edges of H,

$$L_e\left(\tau^{(\ell)}\right) \quad = \quad L_e\left(\tau^{(\ell)}, E''\right) + \sum_{p_\ell} L_e\left(\tau^{(\ell+1)}, E'(p_\ell)\right)$$

and 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 mapping for level-ℓ vertices, which gives $\tau^{(\ell+1)}$. So $L_e(\tau^{(\ell+1)}, E'') = L_e(\tau^{(\ell)}, E'')$ is a deterministic quantity. Note also 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 $\mathbb{E}[L_e(\tau^{(\ell+1)}, E'(p_\ell))] = L_e(\tau^{(\ell+1)}, E'(p_\ell))$ over the random choice of $\tau^{(\ell+1)}(p_\ell)$ as in (13). Finally, by the conditional congestion LP constraints (8), it holds that for any choice of $\tau^{(\ell+1)}(p_\ell)$, $L_e(\tau^{(\ell+1)}, E'(p_\ell)) \leq c_e$.

Thus $L_e(\tau^{(\ell+1)}) - L_e(\tau^{(\ell+1)}, E'')$ is the sum of independent $[0, c_e]$ random variables having mean:

$$L_e\left(\tau^{(\ell)}\right) - L_e\left(\tau^{(\ell)}, E''\right) \quad \leq \quad 16d(\ell+1)\log nd \cdot c_e - L_e\left(\tau^{(\ell)}, E''\right)$$

The inequality uses the fact that $\tau^{(\ell)}$ is good. By a Chernoff Bound (recall that $L_e(\tau^{(\ell+1)}, E'')$ is fixed),

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

this uses the fact that $\ell \leq d$. ■

Applying lemma 4.2 inductively, it follows that:

$$\Pr\left[\tau^{(d+1)} \text{ is good for edge } e\right] \geq 1 - (d+1)/(nd)^4 > 1 - 1/n^4 \tag{14}$$

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.

Observe that the actual load on edge e after the *edge mapping* is the sum of independent $[0, c_e]$ random variables[3] with mean $L_e\left(\tau^{(d+1)}\right)$ which is at most $32d^2\log(nd) \cdot c_e$ when *conditioned* on "$\tau^{(d+1)}$ being good for e". It follows that Pr[Load of $e \geq 64d^2\log(nd) \cdot c_e$] is at most:

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

The first term is upper bounded using Chernoff bound for the edge mapping, and the second term is from (14). Taking 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. 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 care. Using our general framework, this gives poly-logarithmic ratio offline and online algorithms for GraphMap. However, as Räcke decomposition is an intermediate step, we need some more care in the reduction to SingleMap.

5.1 SingleMap on trees

By scaling edge capacities in the substrate graph, we can assume that the workload H is a complete graph K_r with unit demand between every pair of vertices. The substrate graph is a tree $T = (V', E)$ with leaves $V \subseteq V'$, where processes can be mapped only to leaves. There are capacities $c : E \to \mathbb{R}_+$ on edges and $u : V \to \mathbb{R}_+$ on leaves. 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 already determined by an assignment of H-vertices to V.

[3]This is the main reason that flow variables were restricted to "high capacity" edges.

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 $\ell \cdot (r - \ell)$.

Root the tree T at an arbitrary non-leaf 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 non-leaf vertex in T has at most two children (this makes the dynamic program simpler). For any $v \in V'$ let T_v denote the subtree of T rooted at vertex v. Define the following recurrence. For all leaves $v \in V$ and $0 \leq \ell \leq r$, set

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

For any non-leaf vertex $v \in V'$ with children v_1 and v_2, and $0 \leq \ell \leq r$, set

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

where the minimum is over all $0 \leq \ell_1, \ell_2 \leq r$ such that $\ell_1 + \ell_2 = \ell$ and $\ell_1(r - \ell_1) \leq c_{(v,v_1)}$ and $\ell_2(r - \ell_2) \leq c_{(v,v_2)}$. 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 possibilities (at most r many) 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 $M[v, \ell] = \infty$. It is clear that using this recurrence, the value $M[s, r]$ at the root s equals the optimum of the SingleMap instance.

5.2 Offline Algorithm

Here the substrate G is general, and each workload is a complete graphs with unit demands.

The algorithm guesses value $\Lambda \in [\mathsf{Opt}, 2\mathsf{Opt}]$ where Opt is the optimal value– we can try all possibilities. Then we apply the procedure of [16] to substrate G *restricted to edges of capacity at least* $1/\Lambda$, to obtain a Räcke decomposition tree $T(\Lambda)$. Note that the optimal solution uses only edges of capacity at least $1/\Lambda$ in G since $\mathsf{Opt} \leq \Lambda$. The idea behind restricting edges is to ensure that the mapping obtained from the tree only uses high capacity edges of G. 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 the property of Räcke tree $T(\Lambda)$ [4] and guess Λ, the optimal value of this tree instance is at most Λ. So we obtain a mapping on $T(\Lambda)$ having congestion $O(\frac{\log n}{\log \log n}) \cdot \Lambda$. Using the flow template given by Räcke tree $T(\Lambda)$, this yields a splittable-routing solution in G, where:

- The node congestion is $O(\frac{\log n}{\log \log n}) \cdot \Lambda$.
- The edge congestion is $O(\log^2 n \log \log n) \cdot O(\frac{\log n}{\log \log n}) \cdot \Lambda = O(\log^3 n) \cdot \Lambda$.
- Every edge used in this solution has capacity $\geq \frac{1}{\Lambda}$, 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

[4] Every cut in $T(\Lambda)$ has capacity larger than the corresponding cut in G.

of $\mathcal{F}_e$. The total load on any edge $e' \in G$ is the sum of independent $[0, \Lambda\, c_{e'}]$ random variables[5] with mean $O(\log^3 n)\, \Lambda \cdot c_{e'}$. By a Chernoff bound, 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

Recall that the substrate is a general graph G, 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 let R denote the time horizon. We present a poly-logarithmic competitive *randomized* online algorithm. Note that an online algorithm for splittable routing is immediate via Räcke decomposition, given the SingleMap algorithm above and the framework in Section 3. 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 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); 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. Again let $T(\Lambda)$ denote the Räcke decomposition tree from the procedure of [16] applied to substrate G restricted to 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. 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 $\leq \gamma \cdot \Lambda$ where $\gamma = O(\log(nD))$.* Again, using the flow template given by Räcke tree $T(\Lambda)$, the online mapping on $T(\Lambda)$ yields an online splittable-routing solution in G, where:

- 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 Λ. 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 $i \geq 0$, construct Räcke tree $T(\lambda_i)$; all of which are disjoint.

Suppose 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 is $> \gamma \cdot \lambda_i$), we reject this map for H, double Λ and enter phase $i+1$ with H as new workload.

Let ℓ denote the last phase. Let σ_i denote the input sequence received in any phase $i \in [\ell]$. Observe that the optimal value of $\sigma_{\ell-1}$ on 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 congestion induced by σ_i on $T(\lambda_i)$ is at most $\gamma \cdot \lambda_i$, for all $i \in [\ell]$. By the property [16] of the Räcke tree, the congestion induced by σ_i on G is at most

[5] This is because we restricted the splittable-routing solution to high capacity edges, i.e. $c_{e'} \geq 1/\Lambda$.

$\gamma \cdot O(\log^2 n \log\log n) \cdot \lambda_i$. Thus the total congestion on G is $\gamma \cdot O(\log^2 n \log\log n) \cdot \lambda_\ell$; i.e. the competitive ratio is $O(\log^2 n \log\log n \log(nR))$.

Randomized rounding. Finally we obtain an online algorithm for unsplittable routing by randomly rounding the above splittable routing (which has the support property). Note that we perform the rounding for all edges of a workload immediately upon its arrival. If ℓ denotes the last phase of the online algorithm, every edge used in the splittable routing has capacity at least $1/\lambda_\ell$ by the support property. Also for any edge $e' \in G$ at any time, 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 $[0, \lambda_\ell\, c_{e'}]$ random variables with mean as above. By Chernoff bound it follows that for any $e' \in G$ and time t,

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

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

6 Concluding Remarks

We gave 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 focussed on the case that each server has a single resource. We note here that most of our algorithms easily extend to the setting of $r > 1$ resources (eg. CPU, memory, disk, etc.), i.e. 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 ⟨vertex, resource⟩ pair.
- In the algorithm for offline GraphMap (Section 2) the configuration LP has also capacity constraints for each ⟨vertex, resource⟩ pair; the approximation ratio becomes $O\left((\rho_1 + \rho_2) \cdot \frac{\log(nr)}{\log\log(nr)}\right)$ since we use Chernoff bound for $O(nr + n^2)$ events.
- In the online algorithm (Section 3), we work with groundset U consisting of all edges and ⟨vertex, resource⟩ 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 ⟨vertex, resource⟩ pairs. Again the guarantee becomes $O\left(d^2 \cdot \log(ndr)\right)$ due to randomized rounding.
- For complete-graph workloads (Section 5), our algorithm for SingleMap on trees does not extend directly to $r > 1$ resources. However when $r = O(1)$, using standard scaling techniques and a more elaborate dynamic program, we can obtain a *quasi-polynomial* time algorithm.

References

[1] W. P. Adams and T. A. Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 16, pages 43–77, 1994.

[2] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009.

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

[4] Yossi Azar, Bala Kalyanasundaram, Serge A. Plotkin, Kirk Pruhs, and Orli Waarts. Online load balancing of temporary tasks. In *WADS*, pages 119–130, 1993.

[5] N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. Naor, and R. Schwartz. Min-max graph partitioning and small set expansion. In *FOCS (To Appear)*, 2011.

[6] Nikhil Bansal, Kang-Won Lee, Viswanath Nagarajan, and Murtaza Zafer. Minimum congestion mapping in a cloud. In *PODC*, pages 267–276, 2011.

[7] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *STOC*, pages 543–552, 2009.

[8] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $n^{1/4}$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.

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

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

[11] Eden Chlamtac. Approximation algorithms using hierarchies of semidefinite programming relaxations. In *FOCS*, pages 691–701, 2007.

[12] Eden Chlamtac and Gyanit Singh. Improved approximation guarantees through higher levels of sdp hierarchies. In *APPROX-RANDOM*, 2008.

[13] N. Chowdhury, M. Rahman, and R. Boutaba. Virtual network embedding with coordianted node and link mapping. In *INFOCOM*, 2009.

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

[15] Daniel Golovin, Anupam Gupta, Bruce M. Maggs, Florian Oprea, and Michael K. Reiter. Quorum placement in networks: minimizing network congestion. In *PODC*, pages 16–25, 2006.

[16] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *SPAA*, pages 34–43, 2003.

[17] Refael Hassin, Asaf Levin, and Maxim Sviridenko. Approximating the minimum quadratic assignment problems. *ACM TALG*, 6(1), 2009.

[18] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[19] Konstantin Makarychev, Rajsekar Manokaran, and Maxim Sviridenko. Maximum quadratic assignment problem: Reduction from maximum label cover and lp-based approximation algorithm. In *ICALP (1)*, pages 594–604, 2010.

[20] X. Meng, V. Pappas, and L. Zhang. Impact of Data Center Network Architecture on Virtual Machine Placement. In *INFOCOM*, 2010.

[21] Viswanath Nagarajan and Maxim Sviridenko. On the maximum quadratic assignment problem. *Math. Oper. Res.*, 34(4):859–868, 2009.

[22] Harald Räcke. Minimizing congestion in general networks. In *FOCS*, pages 43–52, 2002.

[23] Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[24] Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *STOC*, pages 755–764, 2010.

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

[26] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM CCR*, 38(2):17–29, 2008.

[27] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, 2006.