Topology-Aware VM Migration in Bandwidth Oversubscribed Datacenter Networks

Navendu Jain[1], Ishai Menache[1], Joseph (Seffi) Naor[2*], and F. Bruce Shepherd[3⋆]

[1] Microsoft Research, Redmond, WA
[2] Technion, Haifa, Israel
[3] McGill University

Abstract. Virtualization can deliver significant benefits for cloud computing by enabling VM migration to improve utilization, balance load and alleviate hotspots. While several *mechanisms* exist to migrate VMs, few efforts have focused on optimizing migration *policies* in a multi-rooted tree datacenter network. The general problem has multiple facets, two of which map to generalizations of well-studied problems: (1) Migration of VMs in a bandwidth-oversubscribed tree network generalizes the maximum multicommodity flow problem in a tree, and (2) Migrations must meet load constraints at the servers, mapping to variants of the matching problem – generalized assignment and demand matching. While these problems have been individually studied, a new fundamental challenge is to *simultaneously* handle the packing constraints of server load and tree edge capacities. We give approximation algorithms for several versions of this problem, where the objective is to alleviate a maximal number of hot servers. In the full version of this work [5], we empirically demonstrate the effectiveness of these algorithms through large scale simulations on real data.

1 Introduction

Virtual machine (VM) technology has emerged as a key building block for cloud computing. The idea is to provide a layer of abstraction over resources of a physical server and multiplex them among its hosted VMs. Virtualization provides several benefits such as performance isolation, security, ease-of-management, and flexibility of running applications in a user-customized environment. A typical datacenter comprises tens of thousands of servers hosting VMs organized in racks, clusters or containers e.g., 40-80 servers per rack. These racks are interconnected in a network organized as a spanning tree topology with a high bandwidth oversubscription [4]. As a result, the cost to move data between servers is lowest within the same rack, relatively higher within neighboring racks, and significantly higher between far away ones [4].

⋆ Work done in part while visiting Microsoft Research. Work supported in part by the Technion-Microsoft Electronic Commerce Research Center, by ISF grant 954/11 and by NSERC Discovery and Accelerator Grants.

In a cloud computing setup, the VM load may significantly fluctuate due to time-of-day effects, flash crowds, incremental application growth, and varying resource demand of co-located VMs [11]. This risks the creation of hotspots that can degrade the quality of service (QoS) of hosted applications, e.g., long response delays or low throughput. Therefore, to mitigate hotspots at runtime, cloud platforms provide live migration which transparently moves an entire VM (with its memory/disk state, processor registers, OS, and applications) from an overloaded server to an underloaded one with near-zero downtime, an important feature when live services are being hosted. This VM migration framework represents both a new opportunity and challenge to enable agile and dynamic resource management in data centers [2, 6, 10–13].

While several *mechanisms* exist for live VM migration (e.g., VMware V-Motion, Windows Hyper-V, and Xen XenMotion), there remains a need for optimized, computationally-efficient migration policies. In particular, two key questions need to be answered in any chosen policy:

Q1. Which VMs to migrate from overloaded servers? First, we need to identify which VMs to move from a hotspot so as to reduce server load below a specified threshold. There are various strategies, e.g., selecting VMs till the load falls below the threshold, either in descending order of load and size, or at random. While the former may minimize the number of migrated VMs, the latter may move relatively more VMs while avoiding high migration cost scenarios.

Q2. Which servers to migrate the VMs to? Second, we need to select target servers so as to optimize the placement of selected VMs. In particular, the reconfiguration cost (e.g., bandwidth and latency) to migrate a VM from a hotspot to a target server depends on the network topology between servers. Specifically, in a bandwidth oversubscribed datacenter network, data movement to far nodes risks a long reconfiguration time window, typically proportional to both the network distance and the migrated data volume. Further, VM migrations may interfere with foreground network traffic, risking performance degradation of running applications.

Unfortunately, prior efforts have given little attention to address these challenges. Many cloud systems perform initial provisioning of VMs, but the user needs to detect the hotspot and re-provision VMs to a (likely) different server. Note that determining a new mapping of VMs over physical servers is NP-hard[4]. Several greedy heuristics have been proposed such as first-fit placement of overloaded VMs, applying a sequence of move and swap operations [11], and hottest-to-coldest in which the largest load VM is moved to the coldest server [13]. However, these techniques do not consider the network topology connecting the servers (thereby risking high migration costs in Q2). Others have advocated using stable matching techniques [12] applied to a system of cloud providers and consumers, each aiming to maximize their own benefit. However, the authors assume that VMs are already assigned for migration (thereby skip-

[4] In fact, with general loads, even a single edge network captures the NP-hard knapsack problem, and even unit load versions on trees capture hard instances of edge-disjoint paths [10].

ping Q1) and ignores edge capacity constraints or migration costs in the network connecting the servers. Thus, there is a clear need to develop *automated techniques* to optimize VM migration costs in bandwidth oversubscribed datacenter networks.

The Constrained Migration Model. Servers in a data center are interconnected in an undirected spanning tree network topology with servers as leaf nodes and switches and routers as internal nodes [4]. Each edge in the tree has a capacity measured as bits per time unit.

VMs or jobs (we use these terms interchangeably) are allocated to physical servers with each server typically hosting multiple VMs. Each VM is characterized by three parameters: (i) *transfer size* (typically 1-30 GB); we assume here that these are uniform (e.g., pre-compiled VM images and bounds on allocated RAM) and hence are all normalized to size 1, (ii) *computational load* (e.g., in CPU units); the server load is defined as the sum aggregate of the loads of the VMs hosted on it, and (iii) *value of migration* to prioritize migration of mission-critical VMs. Note that transfer size, load, and value are *independent* parameters.

The set of servers is logically partitioned into *hot* and *cold* servers. For simplicity, we refer to each cold server as a single core having *free* capacity. Exceeding this capacity risks performance degradation of its VMs and hosted applications therein. Each hot server has an *excess* load, quantifying the load reduction needed to meet application QoS.

Our core problem is the *constrained migration problem* (CoMP), formally defined in Section 2. Here we wish to compute a maximal set of hot servers that can be relieved by migrating a subset of their hosted VMs. In our context, in addition to load constraints imposed by server CPU capacity, migration patterns must also obey edge capacities inherent to the topology's bandwidth constraints.

Handling load and size constraints is a nontrivial task. To understand this challenge, we put CoMP in a wider context. First, consider a simplified version in which all server load constraints are ignored and assume that each hot server has already determined a *single* VM which it needs to migrate to alleviate overload. This becomes a maximum disjoint paths problem in a tree, generalizing the maximum matching problem, and is APX-hard even when the edges capacities belong to the set $\{1, 2\}$ [3]. Another special case of CoMP is when all routing constraints are ignored (i.e., tree capacities are set to ∞); i.e., we only have server load constraints. Again, even under the restriction that each server has selected a single job to migrate, this problem instance reduces to a generalized assignment (and maximum demand matching) problem [8, 9]. While these variants have been individually studied, we make initial headway on the new fundamental challenge to *simultaneously* handle packing constraints of server load and tree edge capacities. A longer version of this paper [5] elaborates further on the algorithmic context and its related work.

Algorithmic Contributions. We now give an overview of our results on CoMP. While we are unable to give theoretical bounds for CoMP in its full generality, we obtain approximation algorithms in the following three cases.

1) *Single hot server* (Section 3): We compute a set of VMs at a given hot server for migration to cold servers. Our algorithm either determines that no such set exists (so relieving the hotspot server is not possible) or computes a set of hosted VMs to migrate, that may incur a small additive violation in the load capacities at some of the destination cold servers.

2) *Multiple hot servers* – directed tree approach (Section 4): Our results for the single hot server generalize very nicely to the version of the problem on a directed tree in which the goal is to relieve a maximum number of hot servers. In particular, we provide a bi-criteria guarantee for this case. While the approximate solution for this case does not directly solve CoMP we use it as a building block for our system, called WAVE, briefly described below. A detailed description of WAVE is available in the accompanying technical report [5].

3) *Maximum throughput* – undirected tree approach (Section 5): We consider a maximum throughput relaxation of the problem, in which we aim to maximize the number of VMs that are migrated from hot servers. This version can closely approximate CoMP in scenarios where the set of jobs that are allowed to migrate in each hot server is very small. We extend the integer decomposition approach used in [1] to achieve an 8-approximation for this problem.

Techniques. The CoMP optimization problem is a *packing integer program*. Our solution for the multiple hot server problem is based on a two-phase algorithm. In Phase 1, we first solve a (standard) LP relaxation which fractionally routes the VMs from the hot servers to the cold servers. The key question is how to round this fractional solution without incurring too much loss. Here we devise a new approach - in Phase 2 we reduce ("round" in some sense) this fractional solution to a second LP which is well-structured. In particular, it is defined by a system of totally unimodular constraints, and hence its basic solutions are guaranteed to be integral [7]. Further, we show that its solutions *s*imultaneously relieve the hot servers and satisfy tree edge capacities, at the expense of exceeding the load at each cold server by only a small additive constant.

WAVE - system implementation and evaluation. Based on our directed tree algorithms, we design and evaluate a system called WAVE (Workload Aware VM placEment) for mitigation of hotspots in data centers. WAVE uses a heuristic which iteratively examines hot servers on a rack by rack basis. Specifically, the heuristic invokes our directed tree approximation algorithm by migrating jobs away from hot servers in a single rack (see Figure 1). We iteratively process each rack separately and update the (residual) edge and load capacities when we finish its migrations after each rack iteration.

To evaluate WAVE, we conduct a detailed simulation study based on modeling data center workloads. Our results show that WAVE can efficiently and quickly alleviate single server hotspots and more complex multi-server hotspots via short migration paths, while still scaling to large data centers. The reader is referred to [5] for complete details.

We believe that the problems addressed in this paper open up a new set of challenging theoretical algorithmic questions. These are discussed in [5].

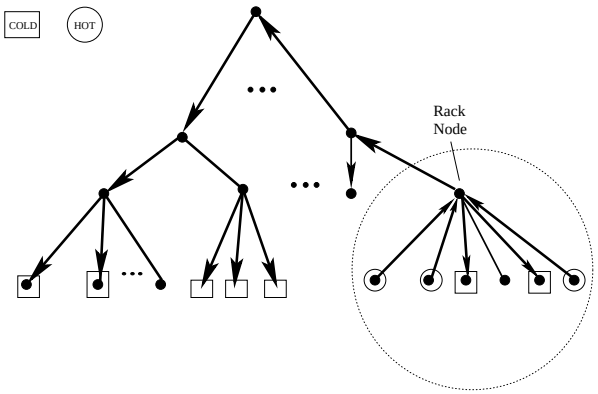

Fig. 1. Tree network topology. Edge orientation is away from hot servers in the specified rack.

2 The Constrained Migration Problem (CoMP)

We now define the optimization problem of VM migration in a bandwidth oversubscribed data center network. The servers are organized in a tree structure denoted by $T = (V, \mathcal{E})$ where V denotes the nodes and edges (or links) are denoted by $e \in \mathcal{E}$. We focus on the case where links are undirected (bidirectional), and state cases where we refer to the directed tree version. In datacenter operations, there may be a temporal budget during which migrations must be completed. To account for this, each link has a capacity denoted by c_e measured as bits per time unit. For practical interest, we logically allocate network capacity for two parts: (i) foreground application traffic and (ii) background VM migrations; we use c_e to denote the latter.

The set of servers is denoted by $\mathcal{K} \subseteq V$. This set is logically partitioned into *hot* and *cold* servers: $\mathcal{K} = \mathcal{K}_{hot} \cup \mathcal{K}_{cold}$. For simplicity, each cold server k denotes a single core having *free* capacity denoted by L_k. Exceeding this capacity risks performance degradation of its VMs and hosted applications therein; a multiplicative constant can be used to model multi-core systems. Multiple resources such as network, disk and memory can be handled similarly but we presently focus on the single resource case. Each hot server h has an *excess* demand denoted by L_h. This quantifies the reduction in load required to meet application QoS.

The set of VMs or jobs are (possibly pre-selected as *subject-to-migration* (STM)) numbered $j = 1, \ldots, N$. Each such job j is characterized by the following parameters: (1) Transfer size s_j, which is normalized to 1 for all VMs. (2) Computational load ℓ_j (e.g., in CPU units) for each job j. (3) The current (hot) server $\mathrm{sv}(j)$ hosting VM j, and a subset $\mathrm{Dest}(j)$, called the *destination set*, of possible (cold) destinations to which j is allowed to migrate. (4) A value (or cost) of migration v_j.

Feasible Solutions. A feasible solution to CoMP specifies a collection $\mathcal{J}$ of jobs, and for each such job there is a target server $k(j) \in \mathcal{K}_{cold}$ for its migration. Obviously j is a job located on some hot server $\text{sv}(j)$ and $k(j) \in \text{Dest}(j)$. For general networks, we would also specify a migration path for such a pair $(j, k(j))$, but this is uniquely determined in a tree topology. Obviously, the total migration of jobs from $\mathcal{J}$ should not exceed the capacity of any edge in T. In addition, if S_h is the set of jobs located at some hot server h, then $\sum(\ell_j : j \in \mathcal{J} \cap S_h) \geq L_h$. Similarly, for each cold server k, $\sum(\ell_j : j \in \mathcal{J} \cap S_k) \leq L_k$.

Objective Functions. The most natural objective function is simply maximizing the number of hot servers to decongest. We call this the *all-or-nothing decongestion version*. Also of interest is the *partial decongestion model*, where the objective is to migrate a maximum weight/number of migrating jobs; we call this the *maximum throughput version* (i.e., one achieves some benefit by partially decongesting servers).

Notation. In the sequel we can always scale link capacities and CPU units so that $s_{min}, \ell_{min} = 1$. We now clarify some notation related to a given undirected graph $H = (V, E)$ with node set V, and edge set E. For any $S \subseteq V(H)$ we denote by $\delta_H(S)$ the set of edges with exactly one endpoint in S. Sometimes we work with a tree T whose node set is also V. For an edge $e \in E(T)$, deleting it from T gives an obvious partition of V into two sets V_1 and V_2. The *fundamental cut* (in H induced by e) consists of $\delta_H(V_1)$. When there is no confusion, we use the notation $\text{Fund}(e)$ to denote the edges in this cut.

3 Relieving a Single Hot Spot (Single Source CoMP)

In this section we consider the single hot server CoMP, i.e., computing a set of jobs at a single hot server h which can be migrated to cold servers, thus relieving the hot server. We describe an approximation algorithm which finds such a set of jobs if one exists, yet might incur a small additive violation in some destination cold server load capacities. Since we have a single hot server, the maximum throughput and all-or-nothing objective functions are equivalent here.

Our starting point is an LP formulation of CoMP. The optimum of the linear program is a *fractional* solution having the property that its value *opt* is an upper bound on the total load of jobs that can be feasibly migrated from h. We then show how to "round" this LP solution to migrate some of these jobs *integrally*. Our rounding process may incur a violation of the total load constraints at some destination cold servers by at most an additive term of ℓ_{max} ($\ell_{max} = \max_i \ell_i$).

Our overall method proceeds as follows. We first solve an LP relaxation. If it does not succeed in reducing the overload of hot server h, then we quit. Otherwise, we use the LP solution to produce a second LP with a nice structure, namely total unimodularity. This implies that all basic solutions for the second LP are integral. We can further prove that the feasible solutions for the second LP still relieve the server h, satisfy tree capacities, and exceed the load at each cold server by at most an additive term of ℓ_{max}.

3.1 Converting a Fractional Migration from a Single Hot Server

We think of our tree T as rooted at node h, i.e., edges are directed "away" from h. This is similar to the orientations in Figure 1, except that here we direct away from only one hot server (as opposed to multiple servers within a rack). We can assume all other leaves of T correspond to cold servers (or else delete them).

We first solve the following natural LP relaxation of CoMP. The maximization objective function guarantees that if there is a feasible solution which decongests h, then the LP optimal value will be at least L_h. That is, it ensures that we fractionally remove enough jobs (at least L_h worth), if that is possible given the constraints (we note that we modify this objective function in our empirical evaluation to incorporate penalties for long migration paths.)

Variables:

- C is the set of cold servers; J is the set of jobs on the hot server.
- $\text{Dest}(j)$ is the set of possible cold servers, for each $j \in J$.
- $x(jk)$ indicates the fractional amount of migration of job $j \in J$ to server $k \in C$.
- $z_j = \sum_{k \in \text{Dest}(j)} x(jk)$ indicates the total fractional migration (in $[0,1]$) of job $j \in J$.

The LP objective: $OPT_{LP1} = \max \sum_{j \in J} \ell_j z_j$
Migration Constraints: for each job j: $\sum_{k \in \text{Dest}(j)} x(jk) \leq 1$
Flow constraints: for each edge $e \in T$: $\sum_{jk \in \text{Fund}(e)} x(jk) \leq c_e$
Load Constraints: for each $k \in C$: $\sum_{j:k \in \text{Dest}(j)} x(jk)\ell_j \leq L_k$
Non-Negativity: $x(jk), z_j \geq 0$

Thus, the LP generates a *fractional migration* vector (x^*, z^*) which aims to decongest the hot server. The components of the solution are: (1) $x^*(jk)$ representing what fraction of job j is migrated to server k (the flow from j to k), and (2) $z_j^* = \sum_k x^*(jk) \leq 1$ representing the total amount of job j on migration. If the solution satisfies $\sum_j \ell_j z_j^* \geq L = L_h$, then the hot server is relieved. We assume x^*, z^* are obtained by an LP solver.

3.2 Multiflows on a Directed Tree

We next recast the fractional migration problem as a *directed multiflow* problem on a tree. This yields another LP - the *Phase 2* LP - which we show has integral optimal solutions (every basic solution is integral). We create the Phase 2 LP from a new directed tree T^* with extra leaves. For each job j, we create a new *job node* j and add a new leaf edge (j, h) from j to the server node h. These edges have capacity 1. We denote by V_J the set of new job nodes in this construction.

We also add new leaves at each cold server. To introduce these, it is convenient to define a *job-edge graph* $H = (V_J \cup \mathcal{K}_{cold}, E_{job})$. For each cold server $k \in \text{Dest}(j)$, we add a *job* edge (j, k) if job j is partially migrated to k in the fractional solution x^*. In this case, if $f = (j, k)$ is such an edge, then we also use ℓ_f to

denote the load ℓ_j of j. Let E_{job} be the resulting set of job edges. These yield the bipartite demand graph.

Note that a feasible (integral) migration of h's jobs corresponds to choosing $M \subseteq E_{job}$ such that: (i) at most one job edge is chosen incident to each $j \in V_J$ (i.e., we do not try to migrate a job twice); (ii) for each edge $e \in T$, the number of job edges "crossing" e (i.e., its fundamental cut $Fund(e)$ in the job-edge graph H) is at most c_e (in other words, the total flow of jobs through e is at most c_e), and (iii) for each $k \in \mathcal{K}_{cold}$, $\sum_{f \in M \cap \delta_H(k)} \ell_f \leq L_k$ (i.e., the total load of jobs migrated to k is at most L_k).

The first two constraints are modeled purely as routing constraints within the tree, i.e., if we choose job edges which have a feasible routing in T^*, then (i) and (ii) hold. The last constraint is different from the first two, since routing a fraction x^*_{jk} of job j to server k induces a load of $\ell_j x^*_{jk}$, and not just x_{jk} which is the induced flow on tree edges (since all sizes are one). This constraint introduces a challenge due to different units for size and load. To address this challenge, we approximately model constraint (iii) as a flow constraint, allowing some additive server overload. To do this, we enlarge T^* with some cold server leaves.

For each cold server k, define its "fractional degree", $f(k)$, to be the total flow (not load) of jobs being migrated to k. We next create new leaf edges at k: $(k,1),(k,2),\ldots,(k,\lceil f(k)\rceil)$, each with capacity one. We call these *bucket leaves* at k. We now redirect the job edges of E_{job} terminating at k to bucket leaf nodes as follows. First, let $f_1, f_2, \ldots, f_p$ be the job edges currently terminating at k, where $f_i = (j_i, k)$. Without loss of generality, assume $\ell_1 \geq \ell_2 \geq \ldots \geq \ell_p$, and consider the fractional amounts $x^*(j_i k)$ that the LP routed from job j_i to server k. We greedily group the f_i's into $\lceil f(k)\rceil$ buckets as follows. Let s be the smallest value such that $\sum_{i=1}^{s} x^*(j_i k) \geq 1$. Then, we remove $f_1, f_2, \ldots, f_s$ from E_{job}, and add instead edges from each j_i to bucket leaf node 1. If the latter sum is strictly larger than 1, then we make two copies of f_s, and the second copy is redirected to leaf node 2. We then proceed to make our buckets $B_1, B_2, \ldots, B_{\lceil f(k)\rceil}$ of job edges in this obvious inductive fashion. Note that the total fractional weight of job edges into each k-leaf node can be viewed as exactly 1, except for the last bin whose total weight is $f(k) - \lfloor f(k)\rfloor$. Figure 2 gives a pictorial example of this operation.

Note that by construction of the multiflow problem, the fractional solution (x^*, z^*) immediately yields a feasible flow in T^*, fractionally migrating the same amount for each job.

Lemma 1. *There is a multiflow in the capacitated tree T^* which routes $x^*(jk)$ for each job j and server k.*

The Phase 2 LP has the following useful property.

Lemma 2. *Any integral solution to the multiflow problem on the expanded directed tree T^* corresponds to a migration which satisfies the above (i), (ii) and*

$$(iii') \qquad \sum_{f \in M \cap \delta_H(k)} \ell_f \leq L_k + \ell_{\max}.$$

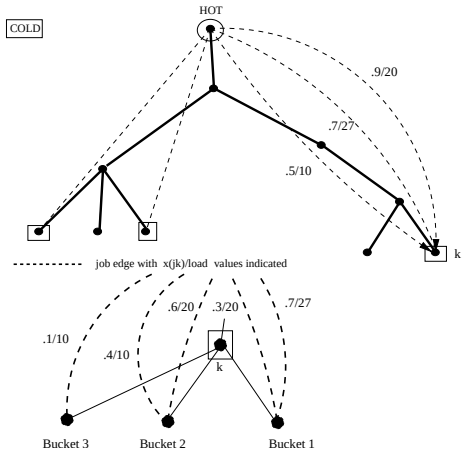

Fig. 2. Construction of buckets at cold server.

3.3 Total unimodularity (TUM) of the routing constraints in T^*

Certain multiflow problems on directed trees have a special structure. Namely, let A be the $\{0,1\}$ matrix whose rows are indexed by directed edges of T^*, and whose columns are indexed by directed paths associated with our job edges E_{job} (where the paths are extended from job nodes to bucket leaves). For a job edge $f = (j, k_r)$ (where k_r denotes some leaf bucket node of cold server k), we put a 1 in row e and column f precisely if e is a directed arc on the path in T^* from j to k_r. It turns out (see [7]) that the resulting matrix A is a network matrix, and hence is *totally unimodular*, i.e., the determinant of every square submatrix is in $\{-1, 0, 1\}$. It follows (cf. [7]) that if $\max w^T y : Ay \leq b, 0 \leq y \leq 1$ has a fractional solution, for some integral capacities $b : E(T^*) \to \mathbf{Z}_+$, then it has an integral basic optimal solution. Since our original solution (x^*, z^*) induces a feasible fractional solution for the multiflow problem on T^*, by taking $w = l$, we must have an integral solution whose objective value is at least $\sum_j \ell_j z_j^*$. That is, we have an almost-feasible (up to (iii') in Lemma 1) integral migration that relieves the hot server. We now combine the pieces to obtain the following.

Theorem 1. *There is a polynomial time algorithm for single-source CoMP in (directed or undirected) trees with the following guarantee. If there is a feasible solution which decongests the single server h, then it finds a decongesting set of jobs which is feasible with respect to tree edge capacities, and violates the load at any cold server by at most an additive amount ℓ_{max}.*

4 All-or-Nothing with Multiple Servers in Directed Trees

The algorithm for resolving a single hot spot works partly due to the fact that the all-or-nothing objective function is equivalent to the maximum throughput

objective function. Namely, by maximizing the number of migrating jobs we can determine whether we succeed in hitting the threshold L_h of server h. In this section we consider the all-or-nothing objective function with multiple hot servers. Hence the complicating knapsack constraints re-enter the picture. We focus on the case where the tree topology is directed, which is also the version used within the system WAVE. We first need to introduce a new multiserver LP which models the relief of a multiple number of hot servers. In particular, its value *opt* is an upper bound on the number of hot servers we can relieve.

The Multiserver LP. We introduce an LP relaxation for multiple hot server migration. It has variables $x(jk)$ and z_j as before, and we also incorporate a variable $m_h \in [0,1]$ for each hot server $h \in \mathcal{K}_{hot}$. Variable m_h measures the (fractional) extent to which we reduce the overload of server h. This is modeled by including the constraints $\sum_{j \in Loc(h)} z_j \ell_j \geq m_h L_h$, and $0 \leq m_h \leq 1$. (Here $Loc(h)$ is the set of jobs on server h available for migration). We then solve this expanded LP with a new objective of maximizing $\sum_{h \in \mathcal{K}_{hot}} m_h$. Note that if *opt* is the optimal value, then this value is an upper bound on the total number of hot servers we could relieve in one round.

Ideally, we would convert a solution to the LP into a valid integral solution which relieves a constant factor $\Omega(opt)$ of servers, with some minimal violation of load constraints at cold servers. Since we consider migration paths in a directed tree, we still inherit a total unimodular structure but there are new difficulties. First, the objective function now uses variables m_h, and these no longer correspond to variables associated with the TUM matrix columns (i.e., the z_j or $x(jk)$ variables). More troubling (theoretically) are difficulties arising from the *all-or-nothing* objective function, i.e., that either a hot server is completely relieved or it is not. The multiserver LP may return solutions in which a large number of m_h's have a very small value, whereas we need to find an all-or-nothing subset where the m_h's are all equal to 1. Currently, our techniques essentially only apply to fractional solutions where we have $\Omega(opt)$ variables $m_h = 1$ (or close to 1). For further details on the theoretical bounds we can obtain through our two-phase LP approach, the reader is referred to [5].

An Iterative Online Heuristic. We employ the algorithmic approach described above within an iterative heuristic for mitigating hotspots across the whole datacenter. This is the basis for our system WAVE, which is reported in [5]. We proceed by addressing hotspot overloads in racks, one by one. Observe that migrations from a rack are *consistent*, in the sense that the resulting orientation on each tree edge is in the same direction. To see this, note that each server which is a child of some rack node is either hot or cold (or neither). Hence, direction of migration along such a leaf edge is determined (upwards from a hot server, or downwards to a cold server); see Figure 1. Moreover, any migrations beyond this rack are obviously all oriented away from the rack node. Thus the migration problem for each rack, has the necessary structure to apply the two-phase LP approach. For each rack, the second optimization yields feasible integral migrations, which are added to a batch scheduling of jobs to be migrated. We then update the tree-capacities and server CPU load capacities used by

this migration, and iteratively select a new rack to mitigate its hot servers. We note that the LP approach allows one to penalize migration paths which are longer. Incorporating such penalty in the objective function leads to significant performance enhancements (more hot servers get migrated and shorter migration paths).

5 Maximum Throughput and *b*-Matched Multiflows

Given the complications inherent to the multiserver LP, we tackle the maximum throughput objective as a (sometimes suitable) alternative. We first note that in the *directed* tree setting, the techniques from the single source algorithm apply directly in the absence of the multiserver LP complications. This is because our technique for binning jobs at the cold servers is oblivious to which hot server the job edge was migrating from. Hence, analogous to Theorem 1, we can migrate LP-OPT-worth of jobs without violating tree capacities, and violating cold server capacities by at most ℓ_{max}.

These techniques do not apply in the undirected setting; we address this now. We formulate maximum throughput CoMP in a slightly more general setting, to emphasize its connection to maximum multiflows in trees (MEDP). In MEDP, we have an edge-capacitated undirected tree $T = (V, E)$, $c : e \to \mathbf{Z}_+$, together with a collection of point-to-point demands $f = uv$ (each demand may also have a profit p_f). The *maximum multiflow problem* (MEDP) asks for a maximum weight (profit) subset of demands that can be simultaneously routed in T without violating edge capacities. A 2-approximation is known for the unweighted version of this problem [3] and 4-approximation for the weighted case [1].

We consider an extension of the above problem where each demand also comes with a load ℓ_f. In addition, each $v \in T$ also comes with a *capacity* (possibly infinite) $b(v)$. A *b-matched multiflow* is a subset of demands F that can be routed in T while satisfying its capacity constraints, and such that for each v: $\sum(\ell_f : f \in \delta(v) \cap F) \leq b(v)$. The problem of finding a maximum b-matched multiflow obviously generalizes MEDP on trees. The reason that it slightly generalizes CoMP on trees is that we no longer have a partition of nodes into hot (supply) and cold (capacitated) servers.

We first establish an 8-approximation for maximum b-matched multiflow, with some additive error in the resource constraints $b(v)$. In fact, we show something stronger, a decomposition result using a technique from [1]. In that work, the authors call a set J of demand edges *k-routable* if when routing all these demands in T, the total flow through any edge is at most kc_e. They prove that any k-routable set J can be partitioned ("colored") into $4k$ sets, each of which is routable (this is the key step to obtaining a polytime 4-approximation for weighted MEDP in trees). With additional work, one can employ their result to obtain a similar decomposition for b-matched multiflows. Our results are the following (further details and proofs appear in [5]).

Theorem 2. *There is an 8-approximation for maximum weighted b-matched mutliflows (and hence maximum throughput CoMP) in undirected trees, if we allow an additive violation of ℓ_{max} at the knapsack constraints for nodes.*

One can also avoid the additive violation of ℓ_{max} with further degradation to the approximation ratio.

Theorem 3. *There is a polynomial time $O(1)$-approximation for maximum weighted b-matched multiflow problem (and hence maximum throughput CoMP) in undirected trees.*

Finally, in the more general *unsplittable flow* setting, each demand f imposes a flow of d_f when it is routed through T's edge capacities. This maximum unsplittable flow problem also has an $O(1)$-approximation on trees [1], assuming the no-bottleneck assumption: $d_{max} \leq c_{min}$. At this point, we have been unable to push these techniques, however, to give constant approximation results in the b-matched unsplittable flow setting.

References

1. C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms (TALG)*, 3(3):27–es, 2007.
2. C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI*, 2005.
3. N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
4. A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *SIGCOMM*, 2009.
5. N. Jain, I. Menache, S. Naor, and F. Shepherd. Topology-aware VM migration in bandwidth oversubscribed datacenter networks. Technical report, May 2012. Available from `http://research.microsoft.com/apps/pubs/?id=162997`.
6. H. A. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Eurosys*, 2009.
7. A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.
8. F. Shepherd and A. Vetta. The demand-matching problem. *Mathematics of Operations Research*, 32(3):563, 2007.
9. D. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1):461–474, 1993.
10. A. Sundararaj, M. Sanghi, J. Lange, and P. Dinda. An optimization problem in adaptive virtual environments. *ACM SIGMETRICS Performance Evaluation Review*, 33(2):6–8, 2005.
11. T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI*, 2007.
12. H. Xu and B. Li. Egalitarian stable matching for VM migration in cloud computing. In *INFOCOM Workshops*, pages 631–636, 2011.
13. Y. Xu and Y. Sekiya. Virtual machine migration strategy in federated cloud. In *Internet Conference*, 2010.