# 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. Finally, we develop a system, called WAVE, to empirically demonstrate the effectiveness of these algorithms through large scale simulations on real data. We use workload traces from a real (not necessarily tree) datacenter topology.

## 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

⋆ 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.

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 inter-connected in a network organized as a spanning tree topology with a high bandwidth oversubscription [6]. As a result, the cost to move data between servers is lowest within the same rack, relatively higher within neighboring racks, and significantly higher when they are further apart [6].

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 [20]. 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 [3, 12, 17, 20, 22, 23].

While several *mechanisms* exist for live VM migration (e.g., VMware VMotion, 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, such as selecting VMs until 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 [20], and hottest-to-coldest in which the largest load VM is moved to the coldest server [23]. 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 [22] 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 skipping Q1) and ignores edge capacity constraints or migration costs in the network connecting the servers. There is a need to further develop *automated techniques* to optimize VM migration costs in bandwidth oversubscribed datacenter networks. As these networks may be large (10's of thousands of nodes), runtime efficiency becomes a nontrivial challenge. This paper presents models and develops several algorithmic approaches to meet these needs.

**The Constrained Migration Model.** We assume servers in a data center to be inter-connected in an undirected spanning tree network topology with servers as leaf nodes, and switches and routers as internal nodes [6]. (See Section 7 for discussion of a much more general setting.) 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* (typ-

[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 [17].

ically 1-30 GB); we assume here that these are uniform (e.g., precompiled 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 in Section 2.1. In particular, we see that several well-studied problems occur as special cases.

**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 bicriteria 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 and in detail in Section 6.

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 [2] 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 [15]. 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.

**Tractability and Real Instance Sizes.**

Our approaches are based on combinatorial rounding techniques for associated linear programming relaxations of CoMP. The scale for real datacenters leads to problem instances with millions of variables and tens to hundreds of thousands of constraints. Since the constraints in our instances are relatively sparse (10-15 nonzeros per column), one would expect that the linear programming (LP) relaxations are solvable by off-the-shelf tools. However, the scale and structure of our Phase 1 integer program does not allow it to be fed directly to a black box solver (as verified by testing with a well-known commercial package). A further challenge arises due to different scale of numerical units in the constraint system: load units for servers (e.g., CPU 0-100%) and VM image size units (e.g., bytes) for flow constraints. Balancing these two types of constraints (in

different units) poses the main challenge in devising combinatorial algorithms.

**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 (not necessarily with a tree topology). 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.

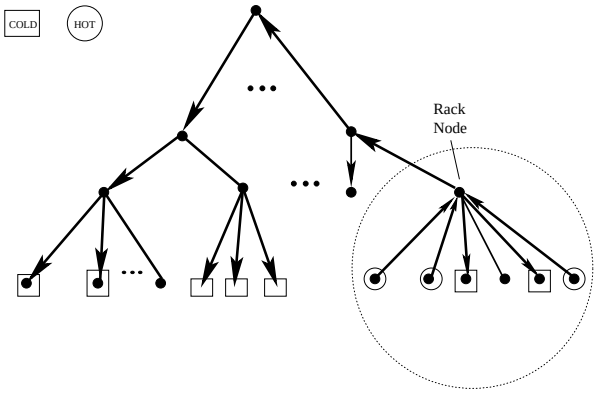


**Fig. 1.** Tree network topology. Edge orientation is away from hot servers in the specified rack.

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 Section 6.

We believe that the problems addressed in this paper open up a new set of challenging theoretical algorithmic questions. These are described as a general optimization framework in Section 7.

## 1.1 Related Systems and Works on VM Migration

The idea of migrating processes or jobs across the network from servers with high loads to servers with lower loads has been an active research area for over 25 years (see, e.g., [8] and references therein). The deployment of large datacenters that support virtualization brings new challenges for live migration of VMs, and has attracted significant attention in both industry and academia. Clark et al. [3] propose the pre-copy approach, which iteratively copies the memory of a VM from the source to destination host before releasing the VM at the source and resuming it at the destination. Hines et al. [9] describe the post-copy migration that defers the transfer of a VMs memory until its processor state has been sent to the target host. Voorsluys et al. [18] study live migration costs on Web 2.0 workloads and conclude that migration overhead is acceptable but cannot be disregarded, especially for quality-of-service sensitive applications. Snowflock enables rapid cloning of stateful VMs but does not consider VM migration [12].

Wide-area VM migration projects [14, 19] have used lazy copy-on reference for moving VM disk state to reduce migration costs over low-bandwidth and high-latency links. Elmroth et al. [4] focus on defining interfaces for initiating and managing VM migration in federated clouds. Wu et al. [21] propose performance models for live migration to predict a VMs migration time given its application behavior and the resources available for migration.

Korupolu et al. [11] investigate the coupled placement of application computation and data amongst available resources, but they only take limited network metrics into account. Sandpiper [20] provides automated black-box, gray-box, and hybrid strategies for VM migration in datacenters. Commercial offerings, such as VMware Distributed Resource Scheduler, use live migration to balance load in response to CPU and memory pressure. However, the scheduler is proprietary and can only be applied to VMs running on VMware hypervisor. All these efforts focus on implementation issues and optimization techniques of migration, but they do not rigorously consider network topology-aware selection of target nodes in datacenter networks.

## 2 The Constrained Migration Problem (CoMP)

We now define the optimization problem of VM migration in a bandwidth oversubscribed data center network. We restrict our focus to tree topologies, although a much broader migration optimization problem is introduced in Section 7.

Servers in our data center 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. Each link has a capacity denoted by $c_e$ measured as bits per time unit. (These are introduce to model a maximum time - time budget - to achieve the migrations.)

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 $L_k$. 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 $\ell_j$ (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 migrating jobs, and for each such job $j$ there is a target server $k(j) \in \mathcal{K}_{cold}$ for its migration. Obviously $j$ is a job located on some hot server $\mathrm{sv}(j)$ and $k(j) \in \mathrm{Dest}(j)$. 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 Fund($e$) to denote the edges in this cut.

### 2.1 Related Problems as Special Cases.

The combinatorial optimization problem CoMP above generalizes several problems in the algorithms literature. For instance, *maximum integer multiflow* (MEDP) in trees can be viewed as a special case. This is seen by having a single job pre-selected at each hot server $h$ and the destination set for this job is to a unique cold server $k$. In addition, we can set $L_k = \infty, L_h = \epsilon$ (for a tiny $\epsilon > 0$) for any cold/hot servers $k/h$. Hence the point-to-point demands for the MEDP instance correspond to our jobs, and routing one such demand corresponds to decongesting the hot server at one of its endpoints. This special case of MEDP is known to be APX-hard even when the tree capacities are $1, 2$ [5]. Conversely, a 2-approximation is known (for general tree capacities) in the cardinality case [5] and a 4-approximation is known in the weighted case [2] (i.e., where each job - hence hot server - has an associated profit or priority $v_j$ for being routed). This latter work is extended in Section 5.

CoMPhas several extra layers of complexity beyond MEDP. First, our jobs are not point-to-point, they may be routed to any of several destinations if Dest($j$) has more than one element. More significantly, we have "decongestion constraints" at the hot servers, and "packing constraints" at the cold servers. Since these constraints are in different units (loads instead of sizes) we cannot model this by simply

adding leaves. Hence on top of our tree-routing capacity constraints, we have these knapsack-type constraints. Moreover, the most natural objective for CoMP is also more complicated. In MEDP, one accrues profit whenever a demand is routed. In CoMPin the all-or-nothing decongestion model, "profit" is obtained at some server only when we succeed to route (migrate) some subset of its jobs which is large enough to decongest it.

Another special case of our problem is when all routing constraints are ignored (i.e., set all tree capacities to $\infty$); then we only have server load constraints. Again, even in the case where each server has selected a single job to migrate, this reduces to a *generalized assignment* problem [16]. Each job $j$ is on one side of a bipartite graph and can be "assigned" to at most one node $i$ on the other side. A profit of $p_{ij}$ is obtained for such an assignment, and the total load of jobs assigned to node $i$ is at most its capacity. The problem also generalizes the multiple knapsack problem.

# 3 Relieving a Single Hot Spot (Single Source CoMP in Trees)

In this section we consider the single hot server CoMP. That is, how to compute a set of jobs at a given single hot server $h$ which can be migrated to cold servers, thus relieving the hot server. In particular, we describe an approximation algorithm which finds a decongesting set of jobs if one exists, but it may have a small additive violation in some destination cold server load capacities.

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, to the tune of an additive term of $\ell_{max}$ ($\ell_{max} = \max_i \ell_i$). Throughout, we assume that all jobs have the same size, i.e., $\forall j,\ s_j = 1$.

Our overall method proceeds as follows. We first solve an LP relaxation. If it does not succeed in reducing 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 means 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 $\ell_{max}$.

### 3.1 Converting a Fractional Migration from a Single Hot Server

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

We first solve the following natural LP relaxation for CoMP. The maximization objective function guarantees that if there is feasible solution which decongests $h$, then the LP optimal value will be at least $L_h$. I.e., it ensures that we fractionally remove enough jobs (at least $L_h$ worth) if it is possible within the constraints. (We tinker with this objective function in our empirical evaluations - see Section 6 - 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 relieves the hot server. The components of the solution are: (1) $x^*(jk)$ which represents how much of job $j$ is migrated to server $k$ (the flow from $j$ to $k$), and (2) $z_j^* = \sum_k x^*(jk) \leq 1$ which represents the total amount of job $j$ which is migrated. The fact that the hot server is relieved corresponds to having $\sum_j \ell_j z_j^* \geq L = L_h$. We assume $x^*, z^*$ are given by any LP algorithm (or solver).

**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 some 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. In order 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 $\ell_f$ to denote the load $\ell_j$ of $j$. Let $E_{job}$ be the resulting set of job edges. These yield the bipartite demand graph $H$.

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 $c_e$)

(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 $s_k = 1$. (This is where different units for size and load make things interesting). Instead, we show how to approximately model constraint (iii) as a flow constraint, if we allow 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 the 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.

This completes the description of $T^*$. The multiflow routing problem on $T^*$ is our Phase 2 LP.

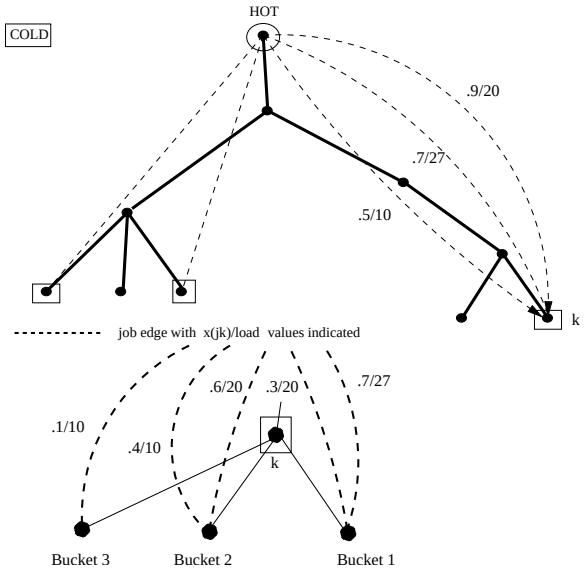


**Fig. 2.** Construction of buckets at cold server.

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 feasible multiflow in $T^*$ which routes $x^*(jk)$ of each job $j$ to each 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*

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

**Proof:** It is only the final claim that we must still argue. To see this, let $M$ denote the set of jobs migrated in a feasible integral flow on $T^*$. Note that $M$ can choose at most one job edge (i.e., job) from any bucket $B_i$ to migrate to $k$. Moreover, for $i > 1$, the size of any job in $B_i$ is at most the size of any job in $B_{i-1}$. Thus, if we choose such a job edge, its load on server $k$ is at most the fractional load

that was induced by job edges in $B_{i-1}$. This happens since the total fractional load induced by edges in $B_{i-1}$ is $\sum_{f \in B_{i-1}} x^*(f)\ell_f$, and this sum is at least the smallest load of a job in $B_{i-1}$, which is as large as any job in $B_i$. Thus, the collective load of jobs chosen from bins $B_2, B_3, \ldots, B_{\lceil f(k) \rceil}$ is upper bounded by the fractional load of $(x^*, z^*)$ on $k$, which is at most $L_k$. Adding in one more job from $B_1$ can thus exceed $L_k$ by an at most $\ell_{max}$.

**Total unimodularity (TUM) of the routing constraints in $T^*$:** 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 [15]) 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. [15]) that if $\max w^T y : Ay \leq b, 1 \geq y \geq 0$ 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 polytime algorithm for single-source CoMP in 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 $\ell_{max}$.*

# 4 All-or-Nothing with Multiple Sources 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 either succeed to hit the threshold $L_h$ or not. In this section we consider the all-or-nothing objective function, and hence the complicating knapsack constraints re-enter the picture.

A second important feature leveraged in the single source case is that migration paths have a consistent orientation away from the hot server. We thus focus on the case where the underlying migration tree has no *orientation conflicts*. In other words, each edge of the tree has a direction, and we only allow job migrations $jk$ for pairs where the path from $j$ to $k$ traverses the edge in the right direction. This generalizes the single hot server case, since in that setting all paths are directed away from the hot server $h$.

The situation is much more complex in the multiserver case, partly due to the difficulty in modelling the choice of which hot servers to decongest. To address this, we must introduce a new *multiserver LP* which models hotspot relief of multiple hot servers. In particular, its value OPT is an upper bound on the number of hot servers we can relieve. Our goal is to ultimately use the LP to find a large (e.g., constant factor of OPT) number of servers which can be relieved.

## 4.1 The Multiserver LP

We introduce an LP relaxation for multiple hot server migration. It has the variables $x(jk)$ and $z_j$ as before, but we also incorporate a variable $m_h \in [0, 1]$ for each hot server $h \in \mathcal{K}_{hot}$. This variables measures the (fractional) extent to which we reduce the overload of $L_h$ at this server. This is modelled by including a constraint

$$\sum_{j \in Loc(h)} z_j \ell_j \geq m_h L_h \tag{1}$$

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 $\sum_{h \in \mathcal{K}_{hot}} m_h$. Note that if OPT is the optimal value, then this 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). To address this issue we use a technique that only guarantees to reduce a congestion by some percentage (we call this $\beta$-decongestion).

More troubling, from the theoretical perspective are difficulties arising from our the *all-or-nothing* objective function. 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 say). We refer to this as an *all-or-nothing* set in an LP solution. In the next section, we show how to convert a fractional migration with such a set into an integral migration. Ending on a positive note, our empirical evaluations show that our Phase 1 LP's often produce non-integral solutions which do posses such large (fractional) all-or-nothing sets - see Section 6.1.

### 4.2 From All-or-Nothing Sets to Valid Migrations

In this section we assume that we have a solution $m_h^*, z_j^*, x^*(jk)$ for the multiserver LP on a directed tree $T$. We assume that this forms an all-or-nothing set. That is, the $m_h^*$ variables belong to $\{0, 1\}$, but the migration variables $x^*(jk)$ themselves may be fractional. (In fact, in the theorem below this is relaxed slightly to only require that the size of the support of $m_h$ is at most $\alpha \sum_h m_h$; if $\alpha = 1$, this corresponds to an all-or-nothing set.) Thus, let $\mathcal{H}$ be a set of hot servers which have been fractionally relieved by the multiserver LP, and let $opt =\mid \mathcal{H} \mid$. We say that a server $h$ is *$\beta$-relieved* if we relieve a total load of at least $\beta L_h$. We show the following.

**Theorem 2.** *Suppose that a fractional migration of value opt is given such that the support of the vector* $(m_h)$ *has size at most* $\alpha \cdot opt$*. Then, we can convert in polynomial-time the solution into an integral migration which* $\beta$*-relieves at least* $\frac{1-2\alpha\beta}{4} opt$ *servers. In particular, we can convert an all-or-nothing migration of opt servers into an integral migration which* $\beta$*-relieves at least* $\frac{1-2\beta}{4} opt$ *servers.*

**Proof:** Once again we convert our problem to a TUM path packing problem on a directed tree. However, in the Phase 2 LP we cannot use variables $m_h$, since they do not correspond to subpaths (or sums of them) within the tree. Instead we use the following *proxy* for $m_h$: $proxy(h) = \sum_{j \in Loc(h)} z_j \frac{\ell_j}{L_h}$. It is easy to see that without loss of generality, in the multiserver LP, the above sum satisfies:

$$proxy(h) \leq 1 + \frac{\ell_{max}}{L_h}. \tag{2}$$

The above just corresponds to not migrating a total load of more than $L_h + \ell_{max}$ from $h$. In the Phase 2 TUM problem we now maximize $\sum_h proxy(h) = \sum_h \sum_j z_j \frac{\ell_j}{L_h}$.

In addition, we perform leaf splitting at every cold server as we did in the maximum throughput case. We now also perform bucketing at each hot server $h$; this is to ensure $\sum_j z_j \frac{\ell_j}{L_h} \leq 2$ in the final solution. This is a device to ensure lots of servers get (partly) decongested. We can view this as bucketing with job sizes $\ell'_j = \ell_j / L_h$, or as bucketing the $\ell_j$'s so we guarantee that the total load of migrated jobs is at most $L_h + \ell_{max}$. Either way, this will guarantee that $proxy(h) \leq 1 + \frac{2\ell_{max}}{L_h}$, where we have used (2). A caveat is that a few of the jobs may get assigned to two buckets and allowing both would destroy our tree structure. To resolve this, we assign it only to the bucket where most of its flow (in terms of $x^*(jk)$ values) was assigned. This could reduce its flow, and hence the $m_h$ values up to 1/2. Hence, we transfer a solution of value $opt/2$ to the Phase 2 LP.

After solving for a basic solution to the Phase 2 LP, by total unimodularity, we have an integral solution whose (proxy) objective is at least $opt/2$. So how many servers did it manage to $\beta$-relieve? Let $X$ be this number, and $Y$ be the number that are not $\beta$-relieved. Since $\mathsf{X}+\mathsf{Y} \leq \alpha opt$, we have $\mathsf{Y} \leq \alpha opt$. Each server in $X$ contributes at most 2 to the proxy objective due to hot server splitting. And the

others obviously contribute at most $\beta$. Hence $2X + \beta Y \geq OPT/2$. Hence $X \geq (OPT/2 - \beta Y)/2$ and since $\mathsf{Y} \leq \alpha opt$, this is at least $\frac{1-2\alpha\beta}{4} OPT$. If we originally had an all-or-nothing solution then $\alpha = 1$. This completes the proof. ■

We employ the algorithmic approach described above within an iterative heuristic for mitigating hotspots across the whole datacenter. See Section 6 for details and comprehensive evaluation of the heuristic.

# 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 $\ell_{max}$.

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 [5] and 4-approximation for the weighted case [2].

We consider an extension of the above problem where each demand also comes with a load $\ell_f$. 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. 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 [2]. 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 ("coloured") 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.

**Theorem 3.** *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 $\ell_{max}$ at the knapsack constraints for nodes.*

**Proof:** Once again there is a natural LP formulation which we solve to get a fractional optimal solution $x^*_f$. In particular, let $H$ be the demand graph, i.e., the edges of $H$ correspond to the demand edges $f$. Then for each edge $e \in T$, the fundamental cut satisfies: $\sum_{f \in Fund(e)} x^*_f \leq c_e$. Also, $\sum_{f \in \delta_H(v)} \ell_f x^*_f \leq b(v)$. We now blow up $x^*$ by some integer $k$ to obtain an integer vector $kx^*$ (this is possible if $x^*$ is a basic solution, e.g., by Cramer's Rule). We can think of $kx^*$ as identifying a multiset $J$ of demand edges: edge $f$ occurs $(kx^*)_e$ times in $J$. Hence each fundamental cut satisfies $|J \cap \text{Fund}(e)| \leq kc_e$.

We adapt an argument from [2] for MEDP on trees; we defer some details to a full version of the paper (in particular, how to cope with non-polynomial size $k$ above). They also create "bins", but they do not have the complication of knapsack constraints at the endpoints of demand edges $f$. For each node $v$, let $J_v = J \cap \delta_H(v)$, and $q_v = |J_v|$. We order (and informally number) the edges of $J$ incident to $v$, say $\ell_1 \geq \ell_2 \geq \ldots \ell_{q_v}$; one should think of each such edge as contributing $1/k$ to the original vector $x^*$. We create bins at $v$ as follows. Bin 1, $B_1 = \{\ell_1, \ell_2, \ldots, \ell_{2k}\}$, $B_2 = \{\ell_{2k+1}, \ell_{2k+2}, \ldots, \ell_{4k}\}$ and so on. We

now try to colour the edges of $J$ so that each colour class consists of a set of demands which can feasibly route in the tree capacities. In addition, we force an extra *bin constraint* that each colour class is allowed to include at most one demand edge from any colour class (for any node $v$). Call this a *strong colour class.*

*Claim.* Any strong colour class $C$ does not violate any knapsack constraint by more than $+\ell_{max}$.

Let $v$ be some node, and set $S = C \cap (J_v - B_1)$ where $B_1$ is $v$'s bin containing the largest load demand edges from $J_v$. Since an element of $C \cap B_i$, $i \in \{2, 3, \ldots, q_v\}$, has load at most that of every element in $B_{i-1}$, we have that $\ell(C \cap B_i) \leq \ell(B_{i-1})/2k$. Thus

$$\ell(S) = \sum_{i=2}^{q_v} \ell(C \cap B_i) \leq \sum_{i=1}^{q_v - 1} \frac{\ell(B_i)}{2k} \leq \frac{\ell(J_v)}{2k} \leq \frac{kb(v)}{2k} = b(v)/2. \quad (3)$$

So $\ell(C \cap J_v) \leq \ell(S) + \ell_1 < (b(v))/2 + \ell_{max}$, establishing the claim.

Now to complete the proof we use the colouring/decomposition result of [2] mentioned before the proof. We add a unit capacity leaf for each bin $B_i$ at each node. Note that our multiset $J$ is now a $2k$-routable set. Clearly it imposes a load of at most $kc_e$ on old edges of $T$, and the new bin leafs carry a load of at most $2k$. Hence we can decompose $J$ into $4(2k)$ colour classes for the extended instance. Since the total profit of the edges in $J$ is $k$ times the profit of $x^*$, at least one of the $8k$ sets in the colouring achieves a profit of at least $\frac{1}{8}$ of the LP. ■

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

**Theorem 4.** *There is a polynomial time $O(1)$-approximation for maximum weighted b-matched multiflow problem (and hence maximum throughput CoMP) in undirected trees.*

**Proof:** We follow the decomposition proof above, and note that (3) implies that the colour class $C$ is actually feasible at $v$, unless $C \cap J_v$ contains precisely one demand whose load $> b(v)/2$. We call such a demand *$v$-fat.* We can now categorize demands according to whether it is fat (for one of its endpoints) or not. Obviously the LP

yields a constant factor of its profit either on fat demands, or non-fat demands. If it is on demands which are not fat, we can apply the decomposition on the LP restricted to those demands. Each colour class is then actually feasible as required.

Otherwise, we restrict attention to the fat demands. First suppose that a constant fraction of the profit is on demands $uv$ which are fat at both ends. In this case, we extend our initial tree to include unit capacity edges at each leaf. Note that the "doubly-fat" edges from $J$ now induce a $2k$-routable set on this tree. Hence we can apply the decomposition method again, and each colour class will be feasible since it contains at most one demand from any leaf.

Finally, we focus on the case where most profit is from "singly-fat" demands. This yields a derived digraph where if $uv$ is a demand edge which is $v$-fat, we include the arc from $u$ to $v$. Now with a further loss of factor 4, one may find a "max cut" in this graph. This determines a set $X$ so that we can focus on those demands that are fat w.r.t. to the nodes in $X$, and not fat w.r.t. nodes in $V - X$. Obviously at nodes in $X$ we can select at most one demand, and this can now be modeled by adding a leaf of capacity 1 as in the doubly-fat case. We now apply the decomposition techniques to the resulting set of demands which is $2k$-routable w.r.t the extended tree. One can think of nodes in $X$ as having a single bin, and there are no large demands for nodes in $V - X$. Hence the decomposition method is again guaranteed to produce feasible colour classes. ■

### 5.1 Non-unit Transfer Sizes: Unsplittable Flow and Extensions

In [2] the authors also consider the extension of MEDP where demands have integer sizes (not just unit sizes $s_j$). This becomes an *unsplittable flow problem* (UFP) on a tree. They give a first constant factor approximation for the resulting maximum UFP on trees when $s_{max} \leq c_{min}$ (the prevalent *no-bottleneck assumption* (NBA)). At this point, we have been unable to push our techniques to give constant approximation results in the $b$-matched unsplittable flow setting.

Such problems fall into the class of so-called *column-restricted packing integer programs* introduced by Kolliopoulos and Stein [10]. These are maximization problems with packing constraints $Ax \leq b$,

where each column $j$ is bi-valued, i.e., all entries are either 0 or some value $v_j$. In addition, it is the NBA which has been widely studied, where $v_j \leq b_i$ for all columns $j$ and rows $i$. Viewed in this generality, each column of the CoMP constraint matrix is tri-valued (each entry in a column $j$ is either 0, $s_j$ or $\ell_j$). It would be interesting to understand the integrality gaps for such problems (with NBA), although as mentioned we do not yet know a result in the special case of UFP on trees.

# 6 Empirical Evaluation

## 6.1 An Iterative Online Heuristic

We employ the algorithmic approach described above within an iterative heuristic for mitigating hotspots across the whole datacenter. We proceed by addressing hotspot overloads in racks, one by one. While the approach from Section 4.2 requires that migrations on each tree edge are *consistent* (i.e., all in the same direction), this is automatically satisfied for migrations from a rack. 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 our methods.

We first apply the Phase 1 LP. If it does not fractionally relieve all hot servers on the rack, we use a binary-search-like routine to try to relieve some smaller set (e.g., half of them). Once we succeed to find an all-or-nothing fractional migration (see comments in next paragraph), we apply the tools from Section 4 to create the Phase 2 LP. 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 select a new rack. We repeat this process as long as the batch of scheduled jobs can be migrated within some target delay (e.g., we used 1 hour).

In our simulations, the $m_h$ values from the Phase 1 LP were not always 1, but fairly large fractions. Hence we worked with these as

a reasonable facsimile for an all-or-nothing solution. We also noted that the proxy objective (2) for the Phase 2 LP had the added advantage that it would tend to drive partial relief at servers, e.g., we see 85% relieved servers in our solutions, whereas in the $m_h$ model, these would contribute nothing to the objective.

In what follows, we describe our simulation results based on datacenter workload traces from a cloud computing cluster running a broad range of interactive and MapReduce/DryadLINQ batch applications. The trace comprises the CPU load over time of each application process running in a VM setup. We begin in Section 6.2 by highlighting some heuristic enhancements which we used in the migration algorithm. We then present the simulation setup in Section 6.3. Section 6.4 focuses on our simulation results, including comparisons to two other algorithms.

### 6.2 Handling Practical Constraints

The datacenter network topology that we examine is that of a multi-rooted spanning tree [7]. Consequently, there are possibly multiple paths from a given hot server to a cold one. While our core algorithm is designed for tree topologies, we exploit the path redundancy as follows: We choose the actual paths by executing a shortest path subroutine (specifically, breadth first search (BFS)) from each hot server to all cold servers. This guarantees that the resulting topology is a tree. Furthermore, if some edges become fully utilized by previous rack migrations, they are eliminated from the graph, thereby allowing the shortest path subroutine to find alternative paths.

Another detail regarding the algorithm implementation is that it does not incorporate "hot-buckets" (we do implement "cold buckets") described in Section **??**. Intuitively, the incorporation of cold buckets guides the integral solution to "follow" the Phase 1 LP. While hot buckets are necessary for obtaining the additive $+\ell_{max}$ performance bounds at destination servers, the system still performed well without this optimization. Occasionally, we observed that some hot servers "over-migrated" without the hot buckets, but this did not appear to be at the expense of other hot servers (see numeric illustration in Section 6.4).

An important heuristic enhancement is to incentivize short migrations paths within the optimization formulation. As we show in Section 6.4, this would lead to *both* increasing the number of relieved servers as well as decreasing the migration delays. The precise details for this improvement are deferred to Section 6.4.

### 6.3 Simulation Setup

**Racks and servers.** We examine our algorithm on a datacenter comprising 8K servers. The servers are organized in racks, where each rack has 40 servers.

**Edge capacities.** As mentioned above, the network topology is a multi-rooted spanning tree with five levels. Foreground application traffic has high network utilization, so that only about 10% of the link capacity can be used for background migration traffic. The available edge capacities which are given in Mbps, are multiplied by a factor $\gamma$ that specifies the worst case migration delay that we are willing to tolerate for each edge. We have set $\gamma$ to 300, so that each edge delay is upper bounded by five minutes, and consequently the overall migration should be bounded by one hour (given the 5-level tree topology).

**Servers loads.** While average DC Utilization is typically around 30% [1], we set the average utilization to be significantly higher, around 50%, to stress test the system as the number of hot servers becomes higher. Accordingly, we proportionally increase the average server load, while using the typical server load distribution and the job-load distribution within each server (roughly, power low distribution for the former, and pareto-distribution for the latter), based on our conversations with datacenter operators.

**Hot and cold servers.** We assume that a server is hot if its current utilization is higher than a parameter HOT-THRESH%. A server is considered cold if its utilization is lower than COLD-THRESH%. Since migrations should not turn cold servers to hot ones, we leave some margins by limiting the utilization at the cold servers after the migration to MAX-LOAD-COLD%. In our experiments, we use HOT-THRESH=80, COLD-THRESH=10, MAX-LOAD-COLD=50, which makes the algorithmic challenge more evident, as the ratio of

hot servers is typically around 20 percent, while only around 12 percent are considered cold servers.

**Algorithm execution.** We order the racks arbitrarily and execute our rack-by-rack algorithm, along with the practical enhancements described above.

### 6.4 Results

As a benchmark, we evaluate our algorithm against the fractional Phase 1 solution which is obtained for each rack. This algorithm is referred to as *Fractional-P1*. We note that this benchmark is of course not a feasible solution, as jobs are allowed to fractionally migrate to different servers. Returning to our WAVE algorithm, we point out that the proxy objective in the Phase 2 LP has a practical advantage – This LP will try to partially relieve some servers as well, whereas under the $m_h$ model (recall $m_h = 1$ precisely if we reduce $h$'s load below HOT-THRESH%) we get no credit for .99-relief. To exploit this advantage, we consider a hot server to be successfully relieved if RELIEF-THRESH percent of the load beyond HOT-THRESH has been migrated. In our experiments, we set RELIEF-THRESH to 85, which means that in the worst case scenario, the target load is violated by only by 3% (in other words, we count the number of successful migrations, assuming that HOT-THRESH is 3% higher).

Before we describe our full results, we present a small illustration of the numbers we obtain for $m_h$. In a particular run, we obtained the following $m$ vectors for the rack migration:

$$m^1 = [1, 1, 0.5672, 1, 1, 1, 1, 1]$$

and

$$m^2 = [1.052, 1.721, 0, 1.692, 1.152, 0.984, 0.875, 1.0093, 0.564].$$

Under the RELIEF-THRESH threshold, the (fractional) benchmark relieved 8/9 of the hot servers, while WAVE relieved 7/9.

We compare our solution to another plausible algorithm which outputs feasible migration. This algorithm simply takes only the integral migrations of the Phase 1 LP, and disregards fractional migrations. We refer to this algorithm as *Integral-P1*, which we a-priori

| The algorithm | % Relieved | STD |
|---|---|---|
| Fractional-P1 | 99.8% | 0.2 % |
| WAVE | 66.5% | 1.2% |
| Integral-P1 | 8.2% | 0.4% |

**Table 1.** Performance Summary. The average and standard-deviation of the ratio between the number of servers relieved by each one of the algorithms, and the total number of hot servers. Results are averaged over 15 runs, where each run handles a different load realization across the datacenter.

thought would yield good results by the nature of the Simplex algorithm. Figure 3 compares the cumulative number of servers relieved for a single run, as a function of the number of racks that have been handled. The WAVE algorithm relieves around 67% of the number of servers predicted by the (infeasible) *Fractional-P1*, while *Integral-P1* is significantly behind with less than 10%. Table 1 summarizes the average ratio of hot servers that are relieved by each one of the three algorithms, which is obtained by dividing the total numbers of servers relieved by the total number of hot servers.

Our simulations indicate that our algorithm significantly outperforms *Integral-P1*, highlighting the benefit of the TUM approach for obtaining integral solutions which "mimic" the fractional solution benchmark.

**Encouraging Shorter Paths: WAVE-SP** We next report a further and substantial improvement to our basic WAVE algorithm, which adds incentives to the optimization process to favor *short migration paths*. Accordingly, the enhanced WAVE algorithm will be referred to as WAVE-SP (i.e., WAVE with shorter paths). WAVE-SP proceeds exactly as WAVE, with the exception that the (maximization) objective function of WAVE-SP includes an additional additive term, namely

$$\sum_h m_h + \alpha \sum_{j \in J} \sum_{k \in \mathrm{Dest}(j)} \frac{x(jk)}{f(dist(j,k))},$$

where $dist(j,k)$ is the distance between the job (hot server) and cold server k, $f(\cdot)$ is a monotone increasing function, and $\alpha$ is a positive constant. We choose $f(x) = x^3$ so that "long" migration are essentially not encouraged. Furthermore, we set $\alpha$ to be small enough

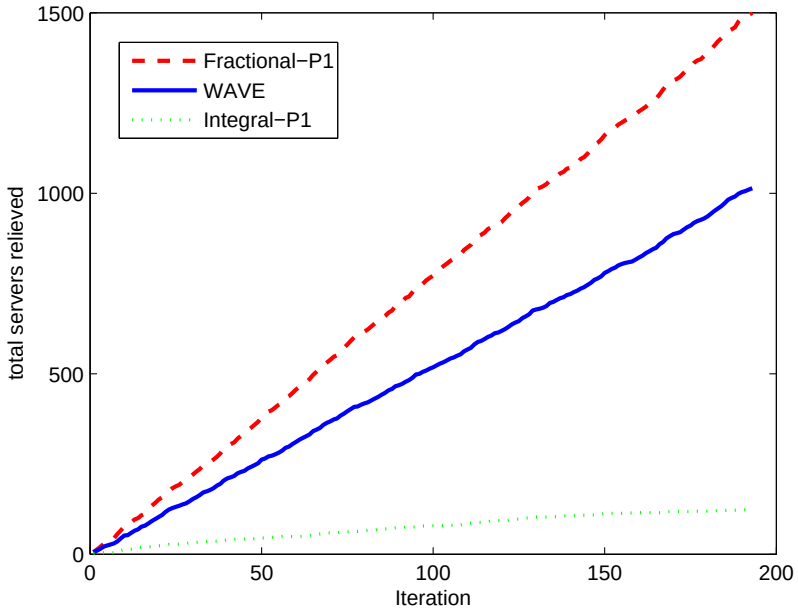


**Fig. 3.** Results of the three algorithms in a typical run. The x-Axis is the iteration number (i.e., the number of racks that have been handled); The y-Axis presents the cumulative sum of the hot servers that have been relieved.

| The algorithm | % Relieved | STD | Hop count | STD |
|---|---|---|---|---|
| WAVE | 66.5% | 1.2% | 4.33 | 0.07 |
| WAVE-SP | 86.0% | 1.3% | 2.52 | 0.05 |

**Table 2.** Comparison between WAVE and WAVE-SP, in terms of (i) the average ratio between the number of servers relieved by each one of the algorithms, and the total number of hot servers; (ii) the average number of hops for the migration. Results are averaged over 15 runs, where each run handles a different load realization across the datacenter.

(specifically, we use $\alpha = 5e^{-4}$), so that $\alpha\|\mathcal{J}\|\|\mathcal{K}\| \max_{j,k} dist(j,k) \approx 1$; that way, the optimization still primarily focuses on relieving hot servers.

This change in the objective function leads to a significant improvement of the *original* objective of relieving hot servers. Specifically, WAVE-SP relieves on average 86.0% of the hot servers. The intuition behind this improvement is that shorter migration paths do not congest the high level edges of the network whose bandwidth is scarce. These edges can be used only when there are really no other alternatives for the migration. We also measure the average migration path length of both WAVE and WAVE-SP (see summary of results in Table 2). As expected, the average path length of WAVE-SP (2.52 hops) is substantially smaller than that of WAVE (4.33 hops). These results indicate that WAVE-SP indeed favors intra-rack migrations, as can be observed from the histogram of the migration hop count for a sample run (Figure 4). This is of course an important improvement on its own, as the migration times decrease proportionally.

# 7 A General Migration Optimization Framework

Jobs (VMs) in data centers may sometimes have non-uniform profiles in terms of their resource usage. The vector bin packing model has thus been proposed to capture this fact [13]. Suppose that we now have $d$ resource types and each job $j$ has a *profile vector* $p_j$; here $p_j(i)$ denotes how much of resource $i$ job $j$ consumes. Previously, the profile vector for job $j$ was simply $(\ell_j, s_j)$ - load and transfer size. In addition, each server $x$ has an *available capacity* $L_x(i)$. If $L_x(i) > 0$,

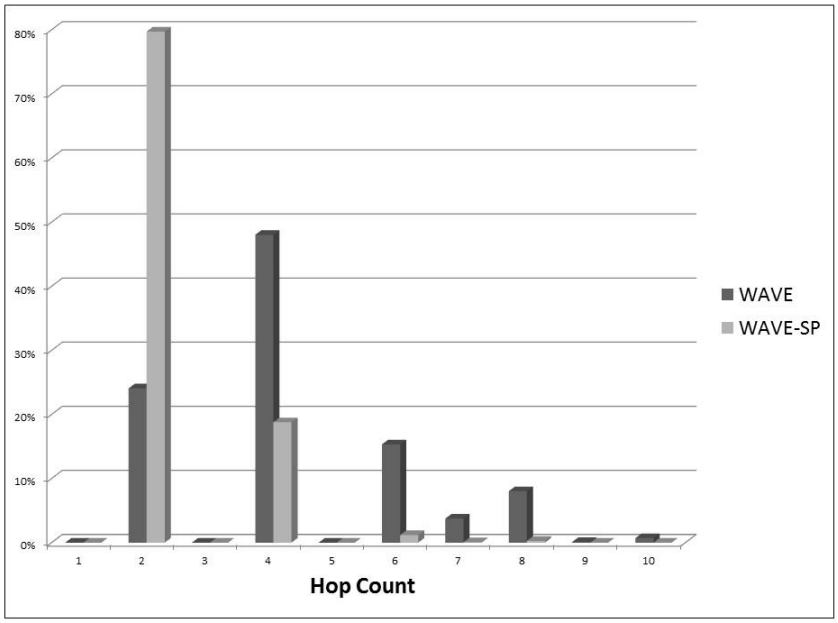


**Fig. 4.** Histogram of the migration path lengths for WAVE and WAVE-SP in a sample run. The x-axis is the hop count (or path length), and the y-axis is the percentage of paths for a given hop count.

then server $x$ has available capacity of resource $i$. If $L_x(i) < 0$, then it is congested in terms of resource $i$ and would like to remove jobs consuming that capacity. In general, the servers are located in some network $G = (V, E)$, e.g., some capacitated undirected graph. (Also more generally, the jobs may have demands but as before, we focus on the unit size/demand case $s_j = 1$).

A *migration vector* now consists of a mapping of some subset of the jobs $j$ to paths $P(j) \in \mathcal{P}_j$, where $\mathcal{P}_j$ are "feasible" paths in $G$, whose endpoints correspond to the end servers associated with job $j$. Let $\mathcal{P}$ denote the set of paths selected by such a mapping. Then $\mathcal{P}$ is *feasible* if $\sum_{P(j)\in\mathcal{P}:e\in P} 1 \leq c_e$ (i.e., it is a feasible integer multiflow in $G$). In addition, we require for each server $x$, and each resource $i$:

$$\sum_{P(j)\in Loc(x)\cap\mathcal{P}} p_j(i) \leq L_x(i)$$

if $x$ has available capacity of resource $i$; otherwise this is a $\geq$ constraint. ($Loc(x)$ is the set of paths which terminate at $x$.) A natural *Migration Optimization Problem* asks for solutions which result in all congestion being alleviated from the network. Exploring this mixed packing and covering problem is an avenue for future work.

## References


1. L. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.
2. 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.
3. 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.
4. E. Elmroth and L. Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *2009 Eighth International Conference on Grid and Cooperative Computing*, pages 253–260. IEEE, 2009.
5. N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
6. 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.
7. A. Greenberg, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 57–62. ACM, 2008.

8. M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems (TOCS)*, 15(3):253–285, 1997.
9. M. R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE*, 2009.
10. S. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. *Integer Programming and Combinatorial Optimization*, pages 153–168, 1998.
11. M. R. Korupolu, A. Singh, and B. Bamba. Coupled placement in modern data centers. In *IPDPS*, 2009.
12. 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.
13. R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing.
14. C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *OSDI*, 2002.
15. A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.
16. D. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1):461–474, 1993.
17. 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.
18. W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom*, 2009.
19. T. Wood, K. Ramakrishnan, J. van der Merwe, and P. Shenoy. Cloudnet: A platform for optimized wan migration of virtual machines. *University of Massachusetts Technical Report TR-2010*, 2, 2010.
20. T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI*, 2007.
21. Y. Wu and M. Zhao. Performance modeling of virtual machine live migration. In *IEEE CLOUD*, 2011.
22. H. Xu and B. Li. Egalitarian stable matching for VM migration in cloud computing. In *INFOCOM Workshops*, pages 631–636, 2011.
23. Y. Xu and Y. Sekiya. Virtual machine migration strategy in federated cloud. In *Internet Conference*, 2010.

# A Pseudo Code

In the following we use the notation Fund($e$) to represent the "fundamental" cut induced by the edge $e$ in $T$. In other words, this set consists of all pairs $jk$ such that $j, k$ are vertices of $T$, and the unique $jk$ path in $T$ uses $e$.

**Algorithm 1** Phase 1 LP

**Input** : Tree Topology $T$, $C$ the set of cold servers
$J$ the set of jobs on the hot server
For each $j \in J$, Dest($j$) is the set of possible cold servers
**Output**: For each $j \in J, k \in C$ migration variable $x(jk)$
total-migration variable $z_j = \sum_{k \in \text{Dest}(j)} x(jk)$
$z_j$ is for ease; expresses the amount of job $j$ migrated in the LP

**LP Objective:** $OPT_{LP1} = \max \sum_{j \in J} l_j z_j$

**for** *each job* $j$, migration constraint **do**
$\sum_{k \in \text{Dest}(j)} x(jk) \leq 1$
**end**
**for** *each edge* $e \in T$, capacity constraint **do**
$\sum_{jk \in \text{Fund}(e)} x(jk) \leq c_e$
**end**
**for** *each edge* $e \in T$, capacity constraint **do**
$\sum_{j:k \in \text{Dest}(j)} x(jk) l_j \leq L_k$
**end**
**for** *each job* $j$, *its* total migration variable **do**
$z_j = \sum_{k \in \text{Dest}(j)} x(jk)$
**end**
$x(jk), z_j \geq 0$

Let $x^*(jk), z_j^*$ be an optimal solution. If $\sum_j z_j^* l_j < L$, then it is not possible to relieve the hot server. QUIT. Otherwise we proceed to the Phase 2 problem, which we formulate as an alternate LP in the next section.

In the *multiserver* case the Phase 1 LP must be adapted slightly. We must naturally incorporate jobs from many servers for the edge capacity constraints on $T$. Namely, if $e = (u, v)$ is an oriented edge of the tree, let $T_u$ denote the subtree containing $u$ after we delete $e$. Similarly, define $T_v$. For each job $j$ at some hot server in $T_u$, and cold server $k \in T_v$, we may include a migration variable $x(jk)$. If we denote by Fund($e$) the set of all such pairs, then the capacity

constraint can again be written as:

$$\sum_{jk \in \text{Fund}(e)} x(jk) \leq c_e.$$

The multiserver LP also has a new objective function

$$\max \sum_h m_h$$

where for each hot server $h$: $0 \leq m_h \leq 1$ and we add a constraint

$$m_h - \sum_{j \in Loc(h), k} x(jk) \frac{l_j}{L_h} \leq 0.$$

We have written this as though all $x(jk)$ migration variables are possible. But using destination sets $\text{Dest}(j)$ we may force some of these to 0, for instance if $jk$ migrates the wrong way on an edge.

## A.1 Pseudo Code for Phase 2 LP

**Algorithm 2** Phase 2 Totally Unimodular LP

**Input** : Phase 1 solution $x^*(jk)$ for each job $j$, cold server $k \in \text{Dest}(j)$
**Output**: For each $j \in J, k \in C$ with $x^*(jk) > 0$, an integer migration variable $x(jk)$
total-migration variable $z_j = \sum_k x(jk)$

**LP Objective:** $OPT_{LP1} = \max \sum_{j \in J} l_j z_j$
**Subject to**
**for** *each job* $k \in C$, *add* **simulated load** *constraints as follows* **do**
  Order jobs $j$ with $x^*(jk) > 0$ in decreasing load size:
  $l_1 \geq l_2 \geq \ldots \geq l_p$
  $f_k \leftarrow \sum_{j \in J} x^*(jk)$ ("fractional degree" of server $k$)
  $f_k^+ \leftarrow \lceil f_k \rceil$ (rounded up degree)
  **for** $i = 1, 2 \ldots, f_k^+ - 1$ **do**
    $o_i \leftarrow \min\{s : \sum_{j=1}^{s} x^*(jk) \geq i\}$ (first point where sum of job flows reach $i$)
  **end**
  $o_{f_k^+} \leftarrow p, o_0 \leftarrow 1$
  **for** $i = 1, 2, \ldots, f_k^+$ **do**
    $B_i \leftarrow \{j : j = o_{i-1}, o_{i-1} + 1, \ldots, o_i\}$ (define Bucket $i$)
    $\sum_{j \in B_i} x(jk) \leq 1$ **bucket** constraint to simulate load at $k$
  **end**
**end**
Add **capacity** and **migration** constraints as in Phase 1 LP
$x(jk), z_j \geq 0$

Solve the new LP. If a simplex (or any basic solution) solver is used, the result will be integral. I.e., $x(jk)$ should be $0-1$ valued, and tells us which jobs to migrate to relieve the hot server.

For the multiserver LP, we can no longer use the $m_h$ variables in a totally unimodular formulation. Instead we drop them altogether and optimize the following proxy objective function.

$$\max \sum_h \sum_{j \in Loc(h), k} x(jk) \frac{l_j}{L_h}.$$

We also add buckets at the hot servers for the technical reason that we do not want to "over-relieve" a hot server (see Theorem 2). Bucketing at $h$ is done the same as for a cold server, except that we must

disallow a job to go to more than one bucket. We order the loads $l_1 \geq l_2 \ldots$ of jobs which were partially migrated in Phase 1. We then iteratively collect them into buckets of total flow 1 as before. If a job is assigned to two buckets, only assign it to the one where more of its flow was sent.