Joint VM Placement and Routing for Data Center Traffic Engineering

Joe Wenjie Jiang*, Tian Lan†, Sangtae Ha*, Minghua Chen⋆, Mung Chiang*
*Princeton University, †George Washington University, ⋆The Chinese University of Hong Kong
{wenjiej, sangtaeh, chiangm}@princeton.edu, tlan@gwu.edu, minghua@ie.cuhk.edu.hk

***Abstract*—Current data centers usually operate under poor utilization due to resource fragmentation. The hierarchical nature of data centers places a limit on the achievable aggregate bandwidth in the backbone. Suboptimal virtual machine placement also introduces unnecessary cross network traffic. In this paper, we solve a joint tenant (*i.e.*, server or virtual machine) placement and route selection problem by exploiting multipath routing capability and dynamic virtual machine migration. These two complementary degrees of freedom: placement and routing, are mutually-dependent, and their joint optimization turns out to substantially improve data center efficiency. We propose (i) an offline algorithm that solves a static problem given a network snapshot, and (ii) an online solution for a dynamic environment with changing traffic. Leveraging and expanding the technique of Markov approximation, we propose an efficient online algorithm that requires a very small number of virtual machine migrations. Performance evaluation that employs the synthesized data center traffic traces, on various topologies and under a spectrum of elephant and mice workloads, demonstrates a consistent and significant improvement over the benchmark achieved by common heuristics used in today's data centers.**

I. Introduction

As demand for data center services continues to rise, efficient traffic engineering within each data center network becomes important, especially under dynamic arrivals of jobs. Optimized placement of jobs on virtual machines helps increase efficiency in node utilization, while optimized routing over multiple paths helps increase efficiency in link utilization. These two optimizations are traditionally carried out separately, yet these two degrees of freedom's effects are clearly coupled, as illustrated in Figure 1. We tackle this combinatorial problem of joint optimization, and develop an approximation algorithm that leverages the specific structures of the joint design problem.

A. Traffic Engineering Inside a Data Center

Cloud services need efficient traffic engineering in their data centers for performance, resource pooling and reliability. There are two common ways to achieve this goal:

Route selection by exploiting multipath capability. Recent data center designs rely on the path multiplicity to achieve scaling of host connectivity. Data center topologies often take the form of multi-rooted spanning trees with one or multiple paths between hosts. However, traditional route selection is neither congestion-aware, nor shortest-path guaranteed, which results in poor bandwidth availability between different parts of the network. Recent developments in data centers allow more sophisticated route selection based on network load on a per-flow basis, *e.g.*, by exploit routing VLANs or OpenFlow [1] protocol. A number of recent work [2], [3] proposed the forwarding protocols to make use of path diversity for data center flows.

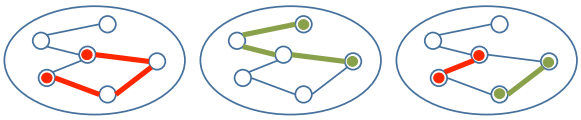


Fig. 1. Illustrating the need for joint design of VM placement and routing. Left: good placement bad routing; middle: good routing bad placement; right: joint optimal placement and routing.

Localizing traffic by flexible VM placement. Modern virtualization based data centers are becoming the mainstream hosting platform for a wide spectrum of application mixtures [4]. An application job usually subscribes a handful of virtual machines (VMs) placed at different hosts that communicate with each other, with different amount of resource requirement for CPU and memory, etc. A number of proposals[5] have been made to improve the agility inside a data center, *i.e.*, any server can be dynamically assigned to any host anywhere in the data center, while maintaining proper security and performance isolation between services. Maximizing the network bisection bandwidth could be viewed as a global optimization problem [6] — servers from all applications must be placed with great care to ensure the sum of their traffic does not saturate any link. However, achieving this level of coordination between changing applications and traffic is difficult in practice [7].

B. Overview of the Paper

Previous work has investigated how to optimally place VMs [6], or select routes for elephant flows in the network [2], [8], but not *both*. Optimizing on any one dimension alone is too restrictive. Suboptimal VM placement introduces unnecessary cross traffic, while oblivious routing even in well-designed fabrics can under-utilize the network resource by several factors [3], [9]. Intuitively, having joint control over both "knobs" provides an opportunity to fully utilize the data center resources. Yet it remains unclear (1) exactly how much improvement can be achieved by jointly leveraging the two mutually-dependent degrees of freedom, and (2) how to computationally efficiently carry out the joint optimization under dynamic job arrivals. These two questions drive this paper.

We focus on the management of network resources by exploiting joint route selection and VM placement. A tenant usually subscribes for each of her VM a certain amount of resource, *e.g.*, CPU and disk, and the uplink/downlink bandwidth SLAs from the data center operators. Fortunately, the operators have control over both where to place to the VMs that meet the resource demand, and how to route the traffic between VMs, at the time when a tenant is admitted. We formalize it as an optimization problem, in which given a sequence of job arrivals, the network operator needs make the routing and placement decisions in order to minimize the network congestion in the long run.

While exploiting both degrees of freedom is undoubtedly appealing to data center operators, it also presents great technical challenge. Solving a static version of the problem, *i.e.*, given all job VM sizes and traffic intensity and optimizing the system performance, is already a challenging problem. The need to make an online decision further complicates the problem, *i.e.*, instead of solving the static problem multiple times and re-optimizing from a scratch which is not only computationally intensive but also cost prohibitive due to a lot of VM migration involved.

In this work, we provide a solution for data center operators, which adapts to changing traffic and applications with *cost-efficient* VM migration schemes. Our contributions are:

- In Section II, we formalize the joint route selection and VM placement problem as a static optimization problem with invariant workloads. In Section III we investigate the combinatorial structure of this problem, and propose an approximation algorithm based on a Markov chain model. We prove that our algorithm can achieve the performance arbitrarily close the global optimum.
- In Section IV, we solve a dynamic version of the joint optimization problem, in which our goal is to optimize the long-term-averaged system performance with time-varying workloads. Building on our solution to the static problem, we develop an incremental online solution which converges to an approximation of the optimum.
- In Section V, we utilize both synthesized and real traffic trace reported from an operational data center to evaluate our algorithms. We show that our solution significantly improves the system performance over some common heuristics used by today's data centers. The benefits are surprisingly stable over different date center topologies, a spectrum of changing workloads and tenant's resource requirements.

Section VI presents related work and Section VII concludes.

II. Joint VM Placement and Routing Optimization

We start with the modeling of a data center network and application jobs, as a *static* problem. We formulate an optimization problem of joint VM placement and routing (VMPR) that aims at minimizing the network congestion.

A. Data Center and VM Model

Consider K jobs from tenants need to be assigned to a data center consisting of M host machines and L links (between machines and switches). Each job $k = 1,\ldots,K$ requires w_k virtual machines. Job k is characterized by $(\mathbf{R}^k, \{\mathbf{S}_{k,i}\}_{i=1,\ldots,w_k})$, where $\mathbf{R}^k$ is an $w_k \times w_k$ matrix with its (i,j) component $\mathbf{R}^k_{i,j}$ representing the traffic load between VMs i and j. The vector $\mathbf{S}_{k,i}$ denotes all physical resource requirements of VM i. For instance, $\mathbf{S}_{k,i}$ has three components when three types of physical resources are considered, *e.g.*, CPU cycles, memory size, and I/O operations respectively. Similarly, the amount of available physical resources provided by host machine $m = 1,\ldots,M$ is given by a vector $\mathbf{H}^m$, *e.g.*, H^m_i is the amount of type-i resource on machine m.

	Jobs and VM
K	Number of tenant jobs, indexed by $k = 1,\ldots,K$.
w_k	Number of VMs required by job k.
$\mathbf{R}^k$	Traffic matrix for communicating VMs of job k.
$\mathbf{S}_{k,i}$	A physical resource requirement vector for VM i of job k.
λ	Job arrival rate.
$1/\mu$	Mean service time of a job.
π_n	Stationary distribution of having n jobs in the system.
	Networks
M	Number of host machines, indexed by $m = 1,\ldots,M$.
$\mathbf{H}^m$	Physical resource vector of machine m.
L	Number of links in a data center network, indexed by $l = 1,\ldots,L$.
G	A graph representation of data center network
V_G	Set of vertices $v_i \in V_G$ on graph G, *i.e.*, , host machines and switches.
E_G	Set of edges $e_l \in E_G$ on graph G, *i.e.*, , links.
S_i	A subgraph with low marginal link costs.
	Optimization
$A^p_{k,i,j}$	A binary decision for routing job k's $i \to j$ traffic on path p.
Y_m	A binary decision for turning on machine m.
$Z^m_{k,i}$	A binary decision for placing VM i of job k on machine m.
r_l	Total traffic load on link l.
f	A feasible configuration, *i.e.*, VM placement and routing.
$\mathscr{F}_n$	A set of all feasible configurations for n VMs.
x^n_f	System objective under configuration f.
x^n_{min}	Optimal system objective under n jobs.
β_n	A positive constant for log-sum-exp approximation.
$q_{f_n \to f_{n+1}}$	Transition rate from state (n, f_n) to state $(n+1, f_{n+1})$.

TABLE I
Summary of key notation

Recently proposed data center architectures allow multi-path routing with commodity layer-2 switches. The topology of a modern data center usually provides rich path diversity between communicating hosts such as VL2, fat-tree and B-cube architectures. We represent a data center by an undirected graph $G = (V_G, E_G)$, where V_G is the set of vertices (*i.e.*, host machines and switches) and E_G is the set of edges (*i.e.*, links). For each job k, we need to find m_k feasible host machines to support its physical resource requirements, as well as a set of path in G to route its traffic among VMs. Each path consists of a set of links that inter-connect switches and host machines.

B. VM Placement and Route Selection

We define $Z^m_{k,i}$ to be a binary variable for placing VM i of job k on host machine m, such that

$$Z^m_{k,i} = \begin{cases} 1 & \text{if VM } i \text{ of job } k \text{ is placed on host } m, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

With this definition, we denote the entire decision space of possible VM placement by the following set

$$\mathscr{Z} \quad = \quad \left\{ \{Z^m_{k,i}\} \mid Z^m_{k,i} \in \{0,1\},\ \sum_{m=1}^{M} Z^m_{k,i} = 1\ \forall (k,i) \right\} \quad (2)$$

where the last equation guarantees that each VM is assigned to exactly one host. Let $Y_m \in \{0,1\}$ be a binary indicator of whether machine m is turned on. For a VM placement to be feasible, the total resource consumption on each host machine cannot exceed its capacity, given that the machine is active.

Traffic load of job k is represented by a matrix $\mathbf{R}^k$, whose (i,j)-th component represents an inelastic traffic load between VMs i and j. To configure routing between VMs i and j, we need to discover a path in G joining the nodes where VMs reside. We define a set of binary routing variables $\{A^p_{k,i,j}\}$ by

$$A^p_{k,i,j} = \begin{cases} 1 & \text{job } k\text{'s } i \to j \text{ traffic is routed on path } p, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

C. Minimizing Network Congestion

Data center operators perform traffic engineering to improve resource utilization, which often can be captured by two terms: (i) network cost that measures the total network congestion, and (ii) node cost which measures the host machine utilization. The former is usually quantified by the average link cost, *e.g.*, convex function of link utilization [10]. The latter is the cost per machine, *e.g.*, energy or capacity cost. We allow operators to freely tune the tradeoff by introducing a weighting factor α. We then formulate the traffic engineering problem as the following constrained optimization:

VMPR

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{L}\sum_l g(r_l/C_l) + \alpha \frac{1}{M}\sum_m Y_m && (4a)\\
\text{subject to} \quad & \sum_k \sum_{i=1}^{w_k} Z^m_{k,i} \cdot \mathbf{S}_{k,i} \preceq Y_m \cdot \mathbf{H}^m,\ \forall m && (4b)\\
& r_l = \sum_k \sum_{(i,j)} \sum_{p \in P(i,j): l \in p} A^p_{k,i,j} R^k_{i,j},\ \forall l && (4c)\\
& \sum_{p \in P(i,j)} A^p_{k,i,j} = 1, \forall k,i,j && (4d)\\
& \{Z^m_{k,i}\} \in \mathscr{Z} && (4e)\\
& \{A^p_{k,i,j}\} \in \mathscr{A}_{\{Z^m_{k,i}\}} && (4f)\\
\text{variables} \quad & A^p_{k,i,j},\ Y_m,\ Z^m_{k,i} && (4g)
\end{aligned}$$

Constraint (4b) models a physical resource constraint. Constraint (4c) captures the total traffic rate on a link, as the sum of traffic intensity over all VM pairs, for all jobs. Constraints (4e) and (4f) ensure feasibility of VM placement and routing. VMPR solves an integer optimization problem, and its decision spaces on VM placement and routing are coupled. Using traffic patterns obtained in measurement study [6], [11], [12], [7], we will take this problem formulation (5) to study joint VM placement and routing for symmetric network architectures in [13], [14], [15], [16], [17].

III. Approximation and Design Approach

The problem **VMPR** is a combinatorial optimization and there is no computationally-efficient solution even in a centralized manner. We leverage the idea of Markov chain approximation in [18] to obtain an approximated solution. The study [18] offers a general framework. We further exploit the unique problem structure of VM placement and routing in data centers to develop a solution that is specially tailored to our needs. In particular, we extend the static problem setting in [18] to a dynamic environment which presents significant implementation challenges. We first study a static problem, and the approach later serves as one of the modules in an efficient solution to the dynamic version of the problem in Section IV. Among the many choices of approximation methods, the one we develop below has the advantage of incorporating specific structures of the data center problem at hand to enhance performance-complexity tradeoff.

A. Approximation Method

Let $f = \{A, Y, Z\}$ be a configuration for the joint VM placement and routing, and $\mathscr{F}$ be the set of feasible configurations defined by constraints (4b)-(4f). For the ease of presentation, we denote x_f as the system objective (4a) given a configuration f. Let each configuration $f \in \mathscr{F}$ be associated with a probability p_f. **VMPR** can be approximated by the following optimization problem, using the approximation technique in [18]:

$$\begin{aligned}
\mathbf{VMPR}(\beta): \min_{\boldsymbol{p} \geq 0} & \sum_{f \in \mathscr{F}} p_f x_f + \frac{1}{\beta} \sum_{f \in \mathscr{F}} p_f \log p_f && (5a)\\
\text{s.t.} & \sum_{f \in \mathscr{F}} p_f = 1. && (5b)
\end{aligned}$$

where β is a large positive constant. As discussed in the next section, this approximation with entropy-term turns out to unique suit our need for accommodating dynamic job arrivals. The optimal solution to $\mathbf{VMPR}(\beta)$ is given by

$$p^*_f(\boldsymbol{x}) = \frac{\exp\left(-\beta x_f\right)}{\sum_{f' \in \mathscr{F}} \exp\left(-\beta x_{f'}\right)},\ \forall f \in \mathscr{F}, \quad (6)$$

and the optimal objective value is

$$-\frac{1}{\beta} \log \left[\sum_{f \in \mathscr{F}} \exp\left(-\beta x_f\right) \right] \approx \min_{f \in \mathscr{F}} x_f$$

With the above approximation method, we obtain an optimal solution to **VMPR**, off by an *entropy* term. If we can time-share among different configurations according to the optimal solution $p^*_f(\boldsymbol{x})$ in (6), then we solve the problem **VMPR** approximately. We thus move on to examine time-sharing strategies that are practical in data centers:

Consider a Markov chain, where each *state* $f \in \mathscr{F}$ is a feasible *configuration* of joint placement and routing decision, *i.e.*, $f = \{A, Y, Z\}$. $\mathscr{F}$ is the set of all possible solutions to problem (5), *i.e.*, $\mathscr{F} = \mathscr{Z} \times \mathscr{A}$. Transitions between two states can mean shuffling of existing VM placement, or routing changes to communicating VMs.

We want the MC to have a desired stationary distribution $p_f^*(\boldsymbol{x})$ in (6). As the system is operated under different configurations, the Markov chain traverses among states and converges to the desired distribution, thus achieving close-to-optimal performance. The particular form of (5b) allows us to restrict our design to time-reversible Markov chains [18], which needs to satisfy the following balance equations:

$$p_f^*(\boldsymbol{x})q_{f,f'} = p_{f'}^*(\boldsymbol{x})q_{f',f}, \forall f, f' \in \mathscr{F}. \tag{7}$$

where $q_{f,f'}$ is the transition rate from state f to f'.

The key design freedom is to build links connecting different states, such that the incurred system cost of state transitions is minimized. In particular, we only allow links connecting two states that can be reached by performing *only* one VM migration. We next show how to implement the strategies in our problem.

B. Offline Algorithm for Snapshot Problem

We next present an offline algorithm that finds the optimal system configuration given a snapshot of the network topology and a fixed number of jobs, leveraging the MC approximation method presented above. The key idea is we allow the system to transit from one state to another by migrating one VM only, with proper transition rate derived from (7). In particular, let

$$q_{f\to f'} \propto \exp^{-1}(-\beta x_{f'}) \tag{8}$$

so the transition rate only depends on the target state. The target state may be the consequence of any one VM migration from the origin configuration. We keep track of the best configuration observed so far and use it as the final solution. The algorithm is illustrated in Figure 2.

In a large data center network, there are an exponential number of neighbor states that can be reached from one state, even if we only allow one VM migration. To reduce the search space, we propose a greedy algorithm that rules out bad possibilities and discovers Δ feasible "good" configurations that can be reached by one VM migration. We present the algorithm in Figure 3. The intuition is that we construct a subgraph from an empty link set and expand it by adding links in an increasing order of marginal cost, *i.e.*, $g'(r_l/C_l)$. A configuration is considered a *candidate target state* if there exists a machine that can host the VM, and is connected to other VMs from the same job. We run the shortest path algorithm in the subgraph to find a possible route between communicating VMs.

To find a close-to-optimal solution in the snapshot problem, the **VMPR-offline** algorithm may require a significant number of VM migrations, which is operationally expensive as moving VMs incurs additional traffic and latency to the service. Repeatedly solving the offline snapshot problem introduces prohibitive computational and operational costs. This motivates us to design a cost-efficient solution that adapts to changing jobs and traffic with a novel MC design.

IV. Online Algorithm for Dynamic Problem

Our goal now is to design a system with cost-efficient VM migrations while achieving close-to-optimal performance. Instead of re-optimizing the problem upon every system change, we utilize such dynamics and design a MC with state transitions that are aligned with VM arrival and departure events.

VMPR-offline
Initialize a feasible system configuration f_0.
$f_{best} \leftarrow f_0$, $x_{best} \leftarrow x_{f_0}$
$t \leftarrow 0$
for $t = 0$ to T
  Randomly pick a VM i
  // Migrate VM i to a new configuration
  $g_i \leftarrow$ VM i's configuration
  Remove VM i from the system, $f_t \leftarrow f_t \backslash g_i$
  $\mathscr{G}_i \leftarrow$ **GreedyAdd**(G, f_t, i)
  $\mathscr{F}_t \leftarrow \{f_t\} \times \mathscr{G}_i$
  Probabilistically choose a configuration $f \in \mathscr{F}_t$ acc. to (8)
  Migrate VM i according to f
  if $x_f < x_{best}$
    $f_{best} \leftarrow f, x_{best} \leftarrow x_f$
  $f_{t+1} \leftarrow f$
end for

Fig. 2. Offline algorithm for VM placement and routing given a fixed number of VMs.

GreedyAdd(G, f_{now}, u)
// Find Δ feasible configuration by migrating VM i given f_{now}
  Initialize $\mathscr{S} = \{S_1, \ldots S_M\}$, where $V(S_i) = \{v_i\}$ and $E(S_i) = \emptyset$
  $\mathscr{G} \leftarrow$ set of feasible configurations for VM i
  $E_G \leftarrow E(G)$
  while $|\mathscr{G}| < \Delta$
    $(v_i, v_j) \leftarrow \arg\min_{l \in E_G} g'(r_l/C_l)$
    $E_G \leftarrow E_G \backslash \{(v_i, v_j)\}$
    Let $v_i \in V(S_{t_1}), v_j \in V(S_{t_2})$
    if $t_1 \neq t_2$ *// Merge two subgraphs S_{t_1}, S_{t_2}*
      $V(S_{t_1}) \leftarrow V(S_{t_1}) + V(S_{t_2}), E(S_{t_1}) \leftarrow E(S_{t_1}) + E(S_{t_2}) + (v_i, v_j)$
    else *// $t_1 = t_2$*
      $E(S_{t_1}) \leftarrow E(S_{t_1}) + (v_i, v_j)$
    $\mathscr{G} \leftarrow \mathscr{G} +$ set of new feasible configurations for VM u in $\mathscr{S}$
  end while
  return $\mathscr{G}$

Fig. 3. **GreedyAdd** algorithm finds a set of locally optimal configurations for VM i. A data center network is represented as a graph $G = (V_G, E_G)$, where each vertex $v_i \in V_G$ denotes a switch or a host machine and each edge $\{v_i, v_j\} \in E_G$ denotes a link. The algorithm iteratively adds links and constructs a set of disjoint low-congestion subgraphs $\mathbf{S} = \{S_1, S_2, \ldots, S_s\}$.

A. Dynamic Markov Chain Approximation

Consider jobs arrive at the system independently at a rate λ, and stay for exponentially long time with mean $\frac{1}{\mu}$. Under this setting, the number of jobs in the system, denoted by n, follows an $M/M/\infty$ queue. Let π_n $(n = 0, 1, \ldots)$ be the stationary distribution of having n jobs in the system. We have

$$\pi_n = \frac{\frac{1}{n!}\rho^n}{\sum_{i=0}^{\infty}\frac{1}{i!}\rho^i} = \frac{1}{n!}\rho^n e^{-\rho},$$

where $\rho = \frac{\lambda}{\mu}$. Let us assume jobs are homogeneous for now. Denote $\mathscr{F}_n$ as the set of all feasible configurations for n jobs. The optimal performance when there are n jobs in the system is defined as

$$x_{\min}^n \triangleq \min_{f \in \mathscr{F}_n} x_f^n,$$

where x_f^n is system objective under configuration $f \in \mathscr{F}$. In this dynamic problem, the optimal long-term averaged system

performance is given by

$$P^* \triangleq \sum_{n=0}^{\infty} \pi_n x_{\min}^n. \tag{9}$$

Applying the log-sum-exp approximation to $x_{\min}^n$, we have

$$x_{\min}^n \approx -\frac{1}{\beta_n} \log\left(\sum_{f\in\mathscr{F}_n} \exp\left(-\beta_n x_f^n\right)\right), \tag{10}$$

where β_n are positive constants controlling the accuracy of the approximation. The long-term average performance then can be rewritten as

$$P \triangleq -\sum_{n=0}^{\infty} \pi_n \frac{1}{\beta_n} \log\left(\sum_{f\in\mathscr{F}_n} \exp\left(-\beta_n x_f^n\right)\right). \tag{11}$$

The following lemma characterizes the gap between P and P^*.

Lemma 1. *For any β_n and $\mathscr{F}_n$, we have*

$$0 \le P - P^* \le \sum_{n=0}^{\infty} \pi_n \frac{1}{\beta_n} \log|\mathscr{F}_n|. \tag{12}$$

For constants $\alpha > 0$, $\theta \ge 0$, γ, and let n_M be the maximum number of VMs that can be accommodated by the system, we have

- *For $\beta_n = \alpha n^\gamma$ and $|\mathscr{F}_n| = \exp(\theta n)$,*

$$0 \le P - P^* \le \frac{\theta}{\alpha} E[n_M^{2-\gamma}],$$

where $E[n_M^{2-\gamma}]$ is the $2-\gamma$ order statistics of n_M.

- *For $\beta_n = \alpha n^\gamma$ and $|\mathscr{F}_n| = n^{\theta n}$,*

$$0 \le P - P^* \le \frac{\theta}{\alpha} E[n_M^{2-\gamma} \log n_M] \le \frac{\theta}{\alpha} E[n_M^{3-\gamma}].$$

Due to space limitation here, all proofs can be found in the online technical report [19].

Implications: 1) As $\beta_n \to \infty$, the approximation in (10) becomes exact. Thus by tuning β_n, one can get as close to the optimal performance as possible. 2) For a practically convenient setting of β_n =constant and $|\mathscr{F}_n| = \exp(\theta n)$, the gap to the optimal performance is bounded and can be made to be arbitrarily small, *i.e.*, $0 \le P - P^* \le \frac{\theta}{\alpha} E[n_M^2]$.

With the approximation of long-term averaged system objective, we next design an optimal MC that achieves the desired performance in its steady state. Let (n, f_n) be a state in which the system accepts n jobs with configuration $f_n \in \mathscr{F}_n$, and p_{n,f_n} be the fraction of time staying in this state. The following lemma characterizes the optimal stationary distribution.

Lemma 2. *P defined in (11) is the optimal value of the following optimization problem*

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & \sum_{n=0}^{\infty} \sum_{f_n\in\mathscr{F}_n} p_{n,f_n} x_{f_n}^n + \sum_{n=0}^{\infty} \sum_{f_n\in\mathscr{F}_n} \frac{1}{\beta_n} p_{n,f_n} \log p_{n,f_n} \\ s.\ t. \quad & \sum_{f_n\in\mathscr{F}_n} p_{n,f_n} = \pi_n, \ n = 1,2,\ldots \end{aligned}$$

Further the optimal solution of the above problem is

$$p_{n,n_f} = \frac{\exp\left(-\beta_n x_{f_n}^n\right)}{\sum_{f\in\mathscr{F}_n} \exp\left(-\beta_n x_f^n\right)} \pi_n, \ \forall f_n \in \mathscr{F}_n, \ n = 1,2,\ldots \tag{13}$$

We next present one construction of such Markov chain and a practical implementation that achieves close-to-optimal stationary distribution.

B. Cost-Efficient VM Migration

We want a scheme that incurs as few VM migrations as possible, while achieving close-to-optimal performance in the long run. The key idea is to only allow state transitions when there are job dynamics, *e.g.*, arrivals and departures, and these transitions involve one VM migration only. In particular, when a new job arrives, the job will be assigned a new configuration, and when there is a job departure, an existing job will be selected to perform the VM migration. We do not change the system configuration when there are no job arrival/departure events. This way we only perform necessary *local* re-optimization to the system, without the need to constantly migrating VMs for every snapshot problem and can significantly reduce the VM migration costs. With the proposed Markov chain structure, we design the transition rates to make sure the resulting Markov chain is time-reversible, and its stationary distribution is optimal as in (13).

1) The Optimal Markov Chain Design: Suppose there are n existing jobs when a new job joins the system. Let $\mathscr{C}$ be the set of local configurations that are available for the new job. We assign the job to the configuration $c \in \mathscr{C}$ such that

$$q_{f_n \to f_{n+1}} = \lambda \frac{\exp\left(-\beta_{n+1} x_{f_{n+1}}^{n+1}\right)}{\sum_{c'\in\mathscr{C}, g_{n+1}=f_n\cup\{c'\}} \exp\left(-\beta_{n+1} x_{g_{n+1}}^{n+1}\right)}. \tag{14}$$

where the system moves from state (n, f_n) to $(n+1, f_{n+1})$, and $f_{n+1} = f_n \cup \{c\}$. That is, the new job is assigned to c with a probability proportional to the exponential of the system performance under the new configuration f_{n+1}. This probability can be computed incrementally without affecting the running VMs.

Upon a job departure, we need to figure out which VMs to migrate. Let $q_{(n+1,f_{n+1})\to(n,f_n)}$ be the transition rate for from state $(n+1, f_{n+1})$ to (n, f_n). We wish to design the transition rates so that the following detailed equations hold: for all (n, f_n) states,

$$p_{n,f_n}^* q_{(n,f_n)\to(n+1,f_{n+1})} = p_{n+1,f_{n+1}}^* q_{(n+1,f_{n+1})\to(n,f_n)} \tag{15}$$

Combining results from (13) and (14), we have

$$\begin{aligned} q_{(n+1,f_{n+1})\to(n,f_n)} \quad = \quad & (n+1)\,\mu \frac{\sum_{f\in\mathscr{F}_{n+1}} \exp\left(-\beta x_f^{n+1}\right)}{\sum_{f\in\mathscr{F}_n} \exp\left(-\beta x_f^n\right)} \times \\ & \frac{\exp\left(-\beta x_{f_n}^n\right)}{\sum_{c'\in\mathscr{C}, g_{n+1}=f_n\cup\{c'\}} \exp\left(-\beta x_{g_{n+1}}^{n+1}\right)} \end{aligned} \tag{16}$$

Lemma 3. *The necessary condition for the designed time-reversible Markov chain with transition rates* (14) *and* (16) *to be implementable is as follows:*

$$\frac{\sum_{f\in\mathscr{F}_n}\exp\left(-\beta x_f^n\right)}{\sum_{f\in\mathscr{F}_{n+1}}\exp\left(-\beta x_f^{n+1}\right)}=\sum_{f_n\in R(f_{n+1})}\frac{\exp\left(-\beta x_{f_n}^n\right)}{\sum_{\substack{c'\in\mathscr{C},\\ g_{n+1}=f_n\cup\{c'\}}}\exp\left(-\beta x_{g_{n+1}}^{n+1}\right)}.$$

where $R(f_{n+1})$ is defined as the set of all states f_n that can lead to f_{n+1} by one step VM migration.

Given that the necessary condition holds, we can implement the Markov chain with the optimal stationary distribution, by following the transition rules in a straightforward way:

- Upon a job arrival, we assign the job to a configuration c with a probability proportional to the exponential of the system performance under configuration $f_{n+1}=f_n\cup\{c\}$. The calculation of this quantity does not require re-shuffling other VMs and the global knowledge.
- Upon a job departure, and suppose the system was in state f_{n+1} before the departure, we choose to switch to a state $f_n\in R(f_{n+1})$ by migrating one remaining job, with probability

$$\frac{\sum_{f\in\mathscr{F}_{n+1}}\exp\left(-\beta x_f^{n+1}\right)}{\sum_{f\in\mathscr{F}_n}\exp\left(-\beta x_f^n\right)}\times\frac{\exp\left(-\beta x_{f_n}^n\right)}{\sum_{c'\in\mathscr{C},g_{n+1}=f_n\cup\{c'\}}\exp\left(-\beta x_{g_{n+1}}^{n+1}\right)}. \quad (17)$$

However, implementing the VM migration upon job departures requires global information of the network and cannot be estimated locally without re-shuffling all VMs. The above observation motivates us to find an approximated MC implementation that is appealing to local implementation.

2) Approximated Markov Chain Promoting Localized Implementation: We next consider a special class of objective function $g(\cdot)$, where g is a linear or piece-wise linear cost function commonly used in traffic engineering problems [10]. Let g'_m be the maximum slope of g, and R_{max} be the maximum VM traffic demand.

Proposition 1. *Define $\Delta=g'_mR_m+\alpha$, then $0\le x_{\min}^{n+1}-x_{\min}^n\le\Delta$. Further, we have*

$$|\mathscr{C}| \ \ge\frac{\sum_{f\in\mathscr{F}_{n+1}}\exp\left(-\beta x_f^{n+1}\right)}{\sum_{f\in\mathscr{F}_n}\exp\left(-\beta x_f^n\right)}\ge\ |\mathscr{C}|\exp\left(-\beta\Delta\right).$$

The result is a straightforward application of the Δ-bound. For large β, the upper bound and lower bound are close and we have the approximation

$$\frac{\sum_{f\in\mathscr{F}_{n+1}}\exp\left(-\beta x_f^{n+1}\right)}{\sum_{f\in\mathscr{F}_n}\exp\left(-\beta x_f^n\right)}\approx|\mathscr{C}|. \quad (18)$$

We apply this approximation to the VM migration probability (17) for job departure and we obtain an approximated MC that enables the local implementation:

$$q_{(n+1,f_{n+1})\to(n,f_n)}=(n+1)\,\mu\times\frac{\exp\left(-\beta\left(x_{f_n}^n+\Delta-\frac{1}{\beta}\log|\mathscr{C}|\right)\right)}{\sum_{c'\in\mathscr{C},g_{n+1}=f_n\cup\{c'\}}\exp\left(-\beta x_{g_{n+1}}^{n+1}\right)}. \quad (19)$$

With such approximation, the Markov chain is easy to implement, requiring no knowledge of global information. The price we pay is that the approximated Markov chain no longer converges to the exact desired stationary distribution as in (13), but to a neighborhood around it. The size of such neighborhood can be characterized using the perturbation approach [20], [21]. The key idea is to treat the approximation as a perturbation to the original Markov chain, and bound the resulting difference in the stationary distribution. It can be shown that

$$\sum_{\forall n,\forall f\in\mathscr{F}_n}\left|\bar{p}_{n,f_n}-p^*_{n,f_n}\right|\le\kappa|E|,$$

where E is the difference between the original transition rate matrix and the approximated (perturbed) matrix, and κ is a conditional number of Markov chain.

C. Online Algorithms

Finally, we present the algorithm that solves the dynamic VMPR problem in Figure 4. The online algorithm consists of two components: job arrival and job departure. Upon a new job arrival, we make local arrangements for the new job without migrating existing jobs and modifying their routes. We reutilize the greedy algorithm presented in Figure 3 to find a set of good neighborhood, and assign the new job to one of the configurations according to the transition probability (14) derived from the optimal Markov chain design. Upon a job departure, we pick one job from the existing tenants and probabilistically migrate it to new machines, according to (19). The candidate job to be migrated are selected among those who create the system bottleneck, *i.e.*, jobs that generate flows on the most congested links. Such heuristic greatly reduces the search space and performs well in practice.

VMPR-online
Upon job k **arrival**:
  $\mathscr{G}_k\leftarrow$ **GreedyAdd**(G,f_{k-1},k)
  $\mathscr{F}_k\leftarrow\{f_{k-1}\}\times\mathscr{G}_k$
  Choose $f\in\mathscr{F}_k$ with probability proportional to $\exp\left(-\beta x_f\right)$
  Accommodate job k and set up routing according to f
  $f_k\leftarrow f$

Upon job k **departure**:
  Find the most-congested edge $e_l\in E(G)$
  Let J be the set of jobs that use edge e_l
  for each job $j\in J$
    Let $\mathscr{F}_j$ be the set of possible configurations by migrating j
    Calculate the transition probability of $f_j\in\mathscr{F}_j$ acc. to (19)
  end for
  $\mathscr{F}_{k-1}\leftarrow\bigcup_{j\in J}\mathscr{F}_j$
  Migrate to configuration $f\in\mathscr{F}_{k-1}$ probabilistically
  $f_{k-1}\leftarrow f$

Fig. 4. Online algorithm for dynamic VM placement and routing problem.

The dynamic VMPR solution is appealing to a practical implementation, since (i) we do not require VM migrations when new jobs arrive and at most one job migration when jobs depart, (ii) the computation of migration probability only requires local information only, *i.e.*, without the need of global knowledge by moving other jobs. The computation of our algorithm upon each job arrival/departure only takes a couple

of seconds even just on a typical laptop for a topology with hundreds of nodes.

V. Performance Evaluation

A. Evaluation Setup

In this section, we evaluate our algorithms on various data center topologies, including clique, fat-tree, HyperX, and BCube. The fat-tree topology in [14] consists of a collection of edge and aggregation switches that form a complete bipartite graph. It offers equal length paths between edge switches. Clique and HyperX in [3] improve scalability of fat-tree by using fewer switches and smaller hop counts. BCube in [17] is a multi-level network architecture where host machines are part of the network infrastructure, i.e., they forward packets on behalf of other hosts. These four topologies have been designed with different goals in mind, and are shown in Figure 9.

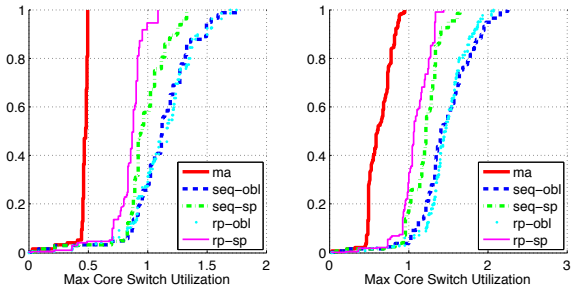
To provide benchmarks for our evaluations, we consider two heuristics for server placement and two heuristics for routing selection, whose combinations give four different baseline strategies.

1) The sequential placement strategy (`seq`) places tenant VMs sequentially in the server stack, *i.e.*, always searching from the lowest-id machine, and tried to place VMs of the same job on the same host machine or in the neighborhood. The `seq` strategy attempts to localize traffic and concentrates host subscription.
2) The random placement strategy (`rp`) randomly picks one or two available host machines for a job. The `rp` strategy ties to spread VMs and their traffic cross network.
3) The shortest path strategy (`sp`) selects a path with minimum end-to-end congestion.
4) The oblivious routing strategy (`obl`) does not consider the congestion level of a path and randomly selects one among the available paths with the shortest hop count.

Our VMPR-online solution (`ma`) presented in the previous section will be compared to these four baseline strategies: `seq-obl`, `seq-sp`, `rp-obl` and `rp-sp`, in various settings of topology, traffic, and application.

We consider the maximum core switch utilization which indicates the utilization of the most congested links connecting core switches, which is often a good estimate of the network bisection bandwidth. We also show the system cost (our objective function) - balancing link cost and node cost.

While data center traffic is quite different from Internet traffic and traffic traces are generally proprietary and unavailable, recent studies on data center traffic measurement do provide us a good characterization on application workload and traffic patterns inside a data center. For example, Ersoz *et al.* [22] characterized the job inter-arrival time of a Web-based service with backend database application as a log-normal distribution, and the job size being also log-normal distributed. Kandula *et al.* [23] reported an empirical observation of flow duration distribution in a 1500-server Microsoft data center over a two-month period. Chen *et al.* [24] showed that the VM resource consumptions in a Google data center can be readily clustered depending on if the job is CPU or memory intensive.



(a) Low traffic load (b) High traffic load

Fig. 5. Performance of online algorithms with synthesized traffic trace on a 128-node fat-tree topology.

Based on the study mentioned above, we construct a synthesized workload emulating the DC traffic. In particular, we use the job inter-arrival time from a log-normal distribution [22], and draw the flow duration from an empirical distribution given in [23]. We pre-define several classes of applications, *e.g.*, CPU-intensive or memory-intensive, each of which has a high consumption on the corresponding resource. We then vary the parameters in these distributions to reflect different workloads in a data center.

We also run our experiment using a real workload from large computing clusters [25]. The workload contains 11 days of activity of cluster machines - job number, arrival time, waiting time, service time, CPU usage, memory usage, etc.

B. Evaluation of Online Algorithms

We next present our simulation results on our online algorithm with the synthesized workload and with the real trace.

Performance of online approximation algorithm. In this scenario, we consider one type of resource and let each VM size requirement be uniformly distributed in [0.2, 0.8]. We vary the job arrival rates to emulate low workload and heavy workload using our synthesized data center workload. We run the `ma` algorithm and four other algorithms using the same traffic trace on a 128-host fat-tree topology.

Figure 5 shows the CDF of maximum core switch utilization under low and heavy traffic load, respectively. In all cases, the `ma` algorithm consistently outperforms other heuristics by as much as 50%, and such improvement is more significant under heavy traffic load. This is because our online approximation algorithm not only takes into account the current job but also future arrivals. Among other four heuristics, `seq-sp` performs well because it optimizes both traffic locality and careful route selection in an intuitive way. However, in the fat-tree topology, the random placement strategy sometimes outperforms the sequential placement strategy due to the very rich path diversity between any pair of hosts provided by fat-tree topology.

Impact of data center topologies. We compare the five algorithms on four typical datacenter topologies: clique, fat-tree, HyperX and BCube. Figure 6 presents the results.

We employ heavy traffic load and assign every link in every topology at the same capacity. The synthesized traffic is used. VM resource requirements are uniformly distributed

between 0.2 and 0.8. We present the topologies in the order of a decreasing degree of path diversity (with an increasing number of switch ports needed). In all topologies with different sizes, our algorithm `ma` shows less congestion in the maximum core switch over other heuristics, with the only exception with the 64-node clique that `seq-sp` performs reasonably well. Since all-pair connectivity between hosts provides good path property and thus has less room for improvement. In general, the performance improvement of `ma` is more significant in topologies with better connectivity and path diversity (unless the topology is completely a clique), especially in the case of large topologies.

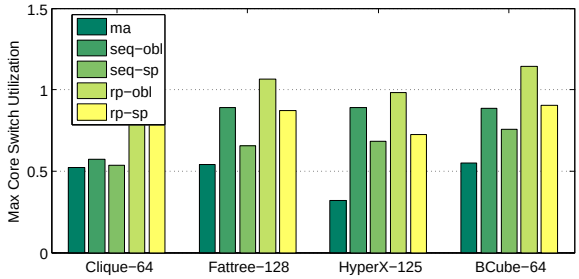


Fig. 6. Comparison of performance on large topologies.

Elephant v.s. mice flows. We next study the impact of flow sizes on the effectiveness of our algorithm. In practice, different applications usually preserve their own traffic patterns, *e.g.*, computation intensive jobs usually generate small but persistent flows that exchange computation results, and file transfers that generate large bursty flows. We define two types of jobs, one that generates elephant flows on the order of 1Gbps, and the other that generates mice flows on the order of 1Mbps. In this experiment, we fix the total amount of workload, *i.e.*, the average traffic rate, while varying the percentage of elephant flows in the network. We run all algorithms with the synthesized traffic on a 125-host BCube topology, and present the performance in Figure 7, under a spectrum of elephant and mice flows mixtures.

A few and very large elephants expectedly amplifiy the worst case congestion. As the fraction of elephant flows increases inside the network, the congestion on the maximum core switch decreases for all algorithms, since the elephant size shrinks given that the total workload is fixed. Our algorithm consistently achieves significant improvement over other heuristics, demonstrating that our algorithm can nicely handle different combination of large and small flows. It is interesting that the improvement margin is quite stable in all fractions of elephants, indicating that our algorithm is insensitive to flow sizes.

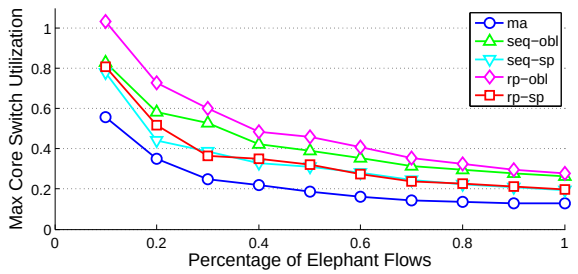


Fig. 7. Performance comparison with varying mix of elephant and mice flows.

Impact of VM size distribution. We next study the impact of different VM size distributions, *i.e.*, the amount of resource required by each VM instance. Different data center applications have their own patterns of resource consumption, *e.g.*, CPU-intensive application request more CPU cycles and similarly memory-intensive applications request more memories. In this experiment we consider only one type of resource. We define the following resource consumption patterns. Applications with uniformly large demand request 100% of host resource for all VMs. Applications with uniformly small demand request 25% of host resource for all VMs. Applications with random demand request an amount uniformly distributed between 0 and 100% for each VM. The bimodal demand is generated from two normal distributions $N(0.3, 0.1)$ and $N(0.7, 0.1)$ with equal probability. All evaluations are performed on a 125-host BCube with the synthesized traffic and results are presented in Figure 8.

Our algorithm `ma` is more effective when the VM size is small and more variant. The sequential placement algorithms (`seq-obl` and `seq-sp`) also work well for uniformly small VM sizes since small VM sizes provide an opportunity to place all VMs on the same machine. Uniformly large VM sizes does not give much room to improve the performance. In both random and bimodal cases, our algorithm improves 50% performance over other heuristics.

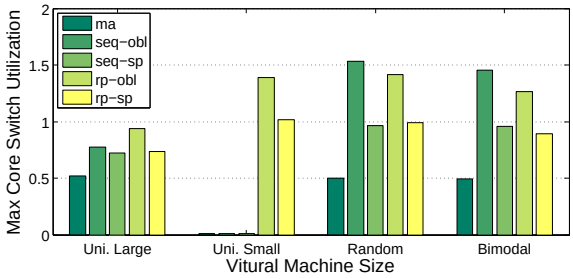
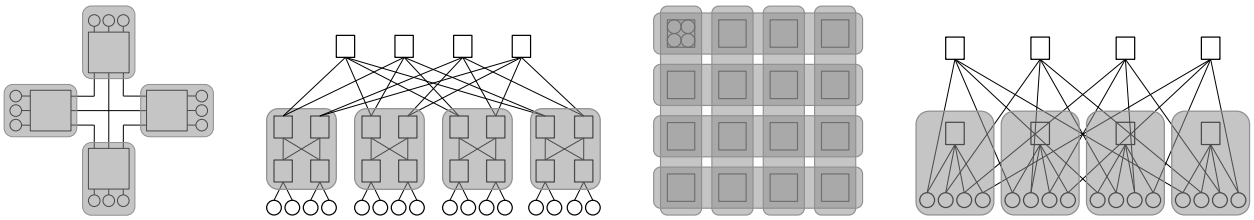


Fig. 8. Impact of different VM sizes.

Real Workload. We evaluate the performance of online algorithms with the real workload obtained from large computing clusters. We extract 4,000 jobs and use its arrival times and flow durations. We fix the topology to fat-tree and use the same parameters used for Figure 5, except that we now vary the intensity of traffic between VMs. Figure 10 shows the performance cost of all online algorithms. As we increase the intensity of traffic between VMs, the overall performance cost increases due to more congestion in the network. As expected, `rp-obl`, which selects the minimum hop count regardless of congestion in the link, becomes worse with higher traffic intensity between VMs. Our algorithm `ma` shows the minimum cost among all tested algorithms.

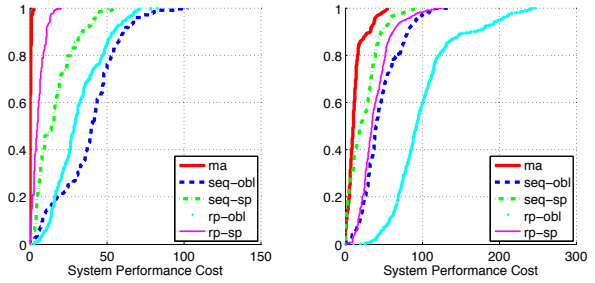
VI. Related Work

Virtual machine placement. In [6], the authors formulated a problem for minimizing total communication distance among VM pairs. The idea is to place VM pairs with heavy traffic among them on host machines with small network cost (which is defined as the number of hops between host machines). In another paper [26], the same authors exploit statistical multiplexing among workload patterns of multiple VMs, and



Clique Fat-Tree HyperX BCube

Fig. 9. Datacenter Architecture



(a) Low VM traffic intensity (b) High VM traffic intensity

Fig. 10. Performance of online algorithms with real work load while varying the intensity of the traffic between VMs.

propose an algorithm for selecting VM combinations with complementary workload patterns.

Scalable data center architecture. Most data centers follow a three tier architecture, *e.g.*, Tree in [13], PortLand in [14], VL2 in [15], and DCell in [16]. Recently, a new multi-level network architecture, BCube, is proposed in [17]. These architectures together with their own routing algorithms have been designed independently with different goals in mind.

Data center traffic measurement. Several papers [6], [11], [22], [24], [23] have provided empirical measurement study on network traffic patterns of commercial cloud services, including Amazon EC2, GFS, MapReduce, and a data warehouse hosted by IBM Global Services. Several trends have been observed: (1) Traffic volumes from VMs are largely uneven. (2) Average traffic per VM is stable at large time-scale. (3) Traffic RTTs have small mean and large variance.

VII. Conclusion

Traditionally, VM placement and routing for data center network are performed separately and the benefits of a joint design are unknown. We solve a joint optimization problem by using the Markov approximation method. Beyond the previous work that only provides a general framework, our new method is specifically tailored to the data center architecture and VM dynamics. Leveraging both synthetic and real traces from operational networks, we demonstrate that our approach is effective, scalable and cost-efficient, compared to common heuristics that miss the opportunity of a joint design.

References

[1] "Openflow," http://www.openflow.org.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2010.

[3] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, "Ensemble routing for datacenter networks," in *ANCS*, 2010.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. ACM Symposium on Operating Systems Principles*, 2003.

[5] C. Kim, "Floodless in seattle: A scalable ethernet architecture for large enterprises," in *ACM SIGCOMM*, 2008.

[6] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *IEEE INFOCOM*, 2010.

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2009.

[8] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "SPAIN: Cots data-center ethernet for multipathing over arbitrary topologies," in *Proc. USENIX NSDI*, 2010.

[9] M. Schlansker, J. Tourrilhes, Y. Turner, and J. Santos, "Killer fabrics for scalable datacenters," in *Proc. ICC*, 2010.

[10] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *IEEE INFOCOM*, pp. 519–528, 2000.

[11] G. Wang and T. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *IEEE INFOCOM*, 2010.

[12] J. P. Srikanth and P. Bahl, "Flyways to de-congest data center networks," in *Proc. SIGCOMM Workshop on Hot Topics in Networking*, 2009.

[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008.

[14] R. Mysore, A. Pamboris, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM*, 2009.

[15] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *ACM SIGCOMM*, 2009.

[16] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *ACM SIGCOMM*, 2008.

[17] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM*, 2009.

[18] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," in *Proc. of IEEE INFOCOM*, (San Diego, CA, USA), 2010.

[19] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," tech. rep., EE, Princeton Univ., 2011, http://www.princeton.edu/ chiangm/vm.pdf.

[20] G. Cho and C. Meyer, "Comparison of perturbation bounds for the stationary distribution of a Markov chain," *Linear Algebra and its Applications*, vol. 335, no. 1-3, pp. 137–150, 2001.

[21] A. Mitrophanov, "The spectral gap and perturbation bounds for reversible continuous-time Markov chains," *Journal of Applied Probability*, vol. 41, no. 4, pp. 1219–1222, 2004.

[22] D. Ersoz, M. S. Yousif, and C. R. Das, "Characterizing network traffic in a cluster-based, multi-tier data center," ICDCS '07, pp. 59–69, 2007.

[23] S. Kandula, S. Sengupta, A. Greenberg, and P. Patel, "The nature of data center traffic: Measurements and analysis," in *IMC*, 2009.

[24] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," Tech. Rep. UCB/EECS-2010-95, EECS, University of California, Berkeley, 2010.

[25] "Logs of real parallel workloads from production systems," http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[26] X. Meng, C. Isci, J. Kephart, L. Zhang, and E. Boulillet, "Efficient resource provisioning in compute clouds via VM multiplexing," in *Proc. ICAC*, 2010.