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
Email: {wenjiej, sangtaeh, chiangm}@princeton.edu, tlan@gwu.edu, minghua@ie.cuhk.edu.hk

***Abstract*—Today's data centers need efficient traffic management to improve resource utilization in their networks. In this work, we study a joint tenant (*e.g.*, server or virtual machine) placement and routing problem to minimize traffic costs. These two complementary degrees of freedom—placement and routing—are mutually-dependent, however, are often optimized *separately* in today's data centers. Leveraging and expanding the technique of Markov approximation, we propose an efficient online algorithm in a dynamic environment under changing traffic loads. The algorithm requires a very small number of virtual machine migrations and is easy to implement in practice. Performance evaluation that employs the real data center traffic traces under a spectrum of elephant and mice flows, demonstrates a consistent and significant improvement over the benchmark achieved by common heuristics.**

I. Introduction

As demand for data center services continues to rise, efficient traffic engineering within each data center network becomes important. 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 structure 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:

Avoiding congestion by intelligent routing. Recent data center designs leverage path multiplicity to achieve the 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 routing is neither congestion-aware, nor shortest-path guaranteed, which results in poor bandwidth 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.

This work has been supported in part by AFOSR grant FA9550-09-1-0643 and ONR N00014-09-1-0449.

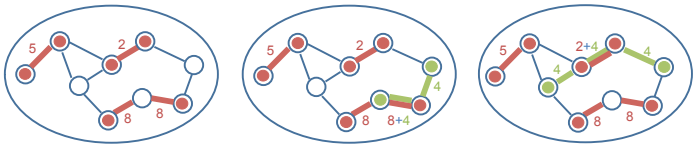


Fig. 1. Illustrating the need for joint design of VM placement and routing: optimizing one on top of the other is sub-optimal. Left: existing VMs and traffic; middle: "good" VM placement with a small distance results in high congestion; right: a "worse" placement allows routing for lower congestion.

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]. A tenant usually subscribes a handful of virtual machines (VMs) placed at different hosts that communicate with one another. The tenant sets up SLAs with the data center operators, *e.g.*, each VM requires a certain amount of resources like CPU, memory, and uplink/downlink bandwidth. 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.

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], [3], 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 [7]. Fortunately, today's data center operators have control over both where to place the VMs that meet the resource requirements, and how to route the traffic between VMs, at the time when a tenant is admitted. We focus on the data center traffic management by exploiting *joint* routing and VM placement. We formalize it as an optimization problem, in which the service operators aim to optimize their network performance (*e.g.*, bisection bandwidth, throughput) and cost (*e.g.*, power consumption).

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 in an *online* fashion. These two questions drive this paper. Our main contributions are:

- In Section II, we formalize the joint routing and VM placement problem and extend the model to an online version that allows dynamic VM arrivals and departures.
- In Section III, we leverage a Markov approximation technique and solve the online joint optimization problem, with the goal of optimizing the *long-term-averaged* performance under changing workloads.
- In Section IV, we utilize synthetic and real traffic trace to evaluate our algorithms. We show that our solution significantly improves the system performance over common heuristics for a wide spectrum of workload distributions.

II. Joint VM Placement and Routing Optimization

We start with the modeling of data center networks for a *fixed* number of tenants and introduce an optimization problem as a joint control of routing and VM placement (VMPR) for optimizing the network performance and cost.

Consider that K jobs (tenants) need to be assigned to a data center consisting of M host machines and L links (between machines and switches). Each job k requires w_k virtual machines. Job k is characterized by $(\mathbf{R}^k, \mathbf{S}_i^k)$, where $\mathbf{R}^k$ is a $w_k \times w_k$ matrix with its (i,j) component $R_{i,j}^k$ representing the traffic load, *e.g.*, the flow size, between VM i and j. The vector $\mathbf{S}_i^k$ denotes physical resource requirements of VM i. For instance, $\mathbf{S}_i^k$ could have three components that capture three types of physical resources such as CPU cycles, memory sizes, and I/O operations, respectively. Accordingly, the amount of physical resource provisioning by host machine m is given by a vector $\mathbf{H}_m$.

A. VM Placement and Route Selection

Placement. For each job k, we need to find a set of machines to host the w_k VMs, as well as a set of paths in G to route the traffic between these VMs. Let $Z_i^k(m)$ be a binary variable, which is equal to 1 if VM i of job k is placed on host m and 0 otherwise. We allow a machine to be turned on or off, *e.g.*, letting $Y_m = 1$ if machine m is turned on, and $Y_m = 0$ otherwise. The feasible decision space for VM placement is characterized by

$$\mathcal{Z} = \left\{ \{Z_i^k(m)\} \;\middle|\; Z_i^k(m) \in \{0,1\},\ \sum_{m=1}^{M} Z_i^k(m) = 1\ \forall (k,i),\ \sum_{k=1}^{K}\sum_{i=1}^{w_k} Z_i^k(m)\cdot \mathbf{S}_i^k \preceq Y_m \cdot \mathbf{H}_m,\ \forall m \right\} \quad (1)$$

where the first equation guarantees that each VM is assigned to exactly one host, and the second equation states that the total resource consumption on a machine should not exceed its total capacity, given that the machine is turned on.

Routing. Today's data center architectures, such as VL2 and B-cube, offer a set of candidate paths for connectivity between any two hosts, in contrast to the traditional spanning-tree based Ethernet routing. In making the routing decisions, we need to select a path connecting two hosts where a pair of communicating VMs are hosted (trivial if the two VMs are co-located). Let $p \in P(i,j)$ be a path that connects VM i and j, and we define a routing variable $A_{i,j}^k(p) = 1$ if VM $i \to j$ traffic of job k is routed on p and $A_{i,j}^k(p) = 0$ otherwise. The routing decision is clearly affected by the routing decision, and the feasible decision space for routing is characterized by

$$\mathcal{A} = \left\{ \{A_{i,j}^k(p)\} \;\middle|\; A_{i,j}^k(p) \in \{0,1\},\ \sum_{p\in P} A_{i,j}^k(p) = 1\ \forall k,(i,j) \right\} \quad (2)$$

B. Optimize Resource Utilization

Data center operators wish to improve the resource utilization inside their networks, which can often be captured by two terms: (i) network cost that measures the network utilization, and (ii) node cost which measures the host machine utilization. The former is usually quantified by link costs that measure the congestion, *e.g.*, using convex-shaped functions that penalize high link utilization [8]. In particular, the network cost is formulated as

$$\text{cost}_{\text{net}} = \sum_l g(r_l/C_l)/L,$$

where $r_l = \sum_{k=1}^{K}\sum_{(i,j)}\sum_{p\in P(i,j): l\in p} A_{i,j}^k(p)\cdot R_{i,j}^k$ is the aggregate traffic rate on link l. We choose the cost function $g(\cdot)$ as suggested in [8], which also applies to other general costs such as the maximum link utilization. The node cost is the operating cost induced by a switch or a machine, *e.g.*, the electricity cost, which can be written as

$$\text{cost}_{\text{node}} = \sum_m Y_m/M.$$

We allow operators to freely strike a balance between the two costs by introducing a weighting factor α. We then formulate the joint VM placement and routing problem as the following optimization:

VMPR$(\mathbf{R}, \mathbf{S})$

$$\begin{array}{ll} \text{minimize} & \text{cost}_{\text{net}} + \alpha \cdot \text{cost}_{\text{node}} \\ \text{variables} & \{Z_i^k(m)\} \in \mathcal{Z},\ \{A_{i,j}^k(p)\} \in \mathcal{A},\ Y_m \in \{0,1\} \end{array}$$

where the constraints are regulated by (1) and (2).

VMPR is a combinatorial optimization and in general there is no efficient solution. In [9], we leverage the idea of Markov chain approximation [10] as a general framework to derive an approximated solution for a given number of jobs. However, in practice, tenants are changing and their jobs may enter and leave the system dynamically. Re-optimizing VMPR for every problem instance is not only computationally expensive, and also introduces high operating costs due to the need for migrating existing VMs as the optimal solution changes. This motivates an online solution that adapts to changing traffic while attaining a good performance-cost tradeoff.

III. Online Solution for VMPR Problem

In this section, we extend the static problem setting to a dynamic environment that presents significant implementation challenges. Among the many choices of approximation methods, the one we develop below has the advantage of incorporating specific structures of the data center network to enhance performance-complexity tradeoff. Due to space limitation, all proofs can be found in the online tech report [9].

A. Approximation Method

We denote $f = \{A, Y, Z\}$ as a configuration for the joint VM placement and routing decision. Let x_f be the system objective (3) under the configuration f. Consider jobs arrive at the system independently at a rate λ, and stay for exponentially long time with mean $1/\mu$. Then the number of jobs n in the system follows an $M/M/\infty$ queue with stationary distribution $\pi_n = \frac{\rho^n e^{-\rho}}{n!}$, where where $\rho = \lambda/\mu$.

Denote $\mathcal{F}_n$ as the set of all feasible configurations for n jobs, defined by (1) and (2). In the dynamic **VMPR**, the optimal long-term averaged system performance is given by

$$P^* \triangleq \sum_{n=0}^{\infty} \pi_n x^n_{\min}, \text{ where } x^n_{\min} \triangleq \min_{f \in \mathcal{F}_n} x^n_f. \tag{3}$$

Here x^n_f is system objective under configuration $f \in \mathcal{F}_n$, and $x^n_{\min}$ is the optimal performance of a particular problem instance with n jobs. After applying the log-sum-exp approximation to the objective as in [10], we have

$$x^n_{\min} \approx -\frac{1}{\beta_n} \log \left(\sum_{f \in \mathcal{F}_n} \exp\left(-\beta_n x^n_f\right) \right), \tag{4}$$

where β_n are positive constants controlling the accuracy of the approximation. As $\beta_n \to \infty$, the approximation in (4) becomes exact. The long-term average performance then can be rewritten as

$$P \triangleq -\sum_{n=0}^{\infty} \pi_n \frac{1}{\beta_n} \log \left(\sum_{f \in \mathcal{F}_n} \exp\left(-\beta_n x^n_f\right) \right). \tag{5}$$

The formal result on the gap between P and P^* is presented in [9]. The underlying implication is that, as $\beta_n \to \infty$, the approximation in (4) becomes exact. By tuning β_n, we are able to get as close to the optimal performance as possible.

We next show how to reach the approximated solution (5) through traversing a time-reversible Markov chain. Let (n, f_n) be a state in which the system accepts n jobs with configuration $f_n \in \mathcal{F}_n$, and p_{n,f_n} be the fraction of time staying in this state. The following theorem characterizes the optimal stationary distribution.

Theorem 1. *P defined in (5) is the optimal value of the following optimization problem*

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

Further the optimal solution of the above problem is

$$p_{n,n_f} = \frac{\exp\left(-\beta_n x^n_{f_n}\right)}{\sum_{f \in \mathcal{F}_n} \exp\left(-\beta_n x^n_f\right)} \pi_n, \ \forall f_n \in \mathcal{F}_n, \ n \tag{6}$$

As such, we obtain the optimal solution (3), off by an *entropy* term. If we can time-share among different configurations according to the optimal solution $\boldsymbol{p}^*$ in (6), then we solve the online problem approximately. We next move on to examine practical time-sharing strategies in data centers.

B. Cost-Efficient Placement and Routing

Consider a Markov chain (MC), where each *state* (n, f_n) means that the system accepts n jobs with configuration f_n. Transitions between two states can mean a VM arrival/departure, shuffling of existing VM placement, and routing changes to communicating VMs. We want the MC to achieve the desired stationary distribution $\boldsymbol{p}^*$ in (6), while incuring as few VM migrations as possible. The key idea is to only allow state transitions in the presence of 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-optimizations to the system, without the need to constantly migrating VMs for every problem instance and can significantly reduce the VM migration costs. We first show how to construct the MC optimally, and then introduce an approximated MC that is appealing to practical implementation.

Optimal Markov Chain Design. The particular form of (6) allows us to restrict our choices to time-reversible Markov chains [10], which need to satisfy the following balance equations:

$$p^*_{n,f_n} q_{(n,f_n)\to(n+1,f_{n+1})} = p^*_{n+1,f_{n+1}} q_{(n+1,f_{n+1})\to(n,f_n)} \tag{7}$$

where $q_{(n,f_n)\to(n+1,f_{n+1})}$ is the transition rate from state (n, f_n) to $(n+1, f_{n+1})$.

Suppose n jobs exist and a new job arrives. Let $\mathcal{C}$ be the set of local configurations that are available for the new job. We assign the job to the configuration $c \in \mathcal{C}$ such that

$$q_{f_n \to f_{n+1}} = \lambda \frac{\exp\left(-\beta_{n+1} x^{n+1}_{f_{n+1}}\right)}{\sum_{c' \in \mathcal{C}, g_{n+1} = f_n \cup \{c'\}} \exp\left(-\beta_{n+1} x^{n+1}_{g_{n+1}}\right)}. \tag{8}$$

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 how to update the system configuration, *e.g.*, by migrating

existing VMs. Combining (6) and (8), we have

$$q_{(n+1,f_{n+1})\to(n,f_n)} = (n+1)\,\mu \frac{\sum_{f\in\mathcal{F}_{n+1}} \exp\left(-\beta x_f^{n+1}\right)}{\sum_{f\in\mathcal{F}_n} \exp\left(-\beta x_f^{n}\right)} \times \frac{\exp\left(-\beta x_{f_n}^{n}\right)}{\sum_{c'\in\mathcal{C}, g_{n+1}=f_n\cup\{c'\}} \exp\left(-\beta x_{g_{n+1}}^{n+1}\right)} \quad (9)$$

Theorem 2. *The necessary condition for the designed time-reversible Markov chain with transition rates* (8)-(9) *is:*

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

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: (i) Upon a job arrival, we assign the job to a configuration c with a probability according to (8). The calculation of this quantity does not require the global knowledge and shuffling existing VMs. (ii) 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 a probability according to (9). However, implementing the VM migration upon job departures requires global information and cannot be estimated locally without moving existing VMs. This motivates us to find an approximated MC implementation that is appealing to local implementation.

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 [8]. Let g'_m be the maximum slope of g, and R_{max} be the maximum VM traffic demand. Denote $\Delta = g'_m R_m + \alpha$, then $0 \le x_{\min}^{n+1} - x_{\min}^{n} \le \Delta$. Further, the following quantity is bounded by

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

The result is a straightforward application of the Δ-bound. For large β, the upper bound and lower bound are close and we apply the approximation $\mathcal{C}$ to the VM migration probability (9) for job departure. We thus obtain an approximated MC that enables the local implementation:

$$q_{(n+1,f_{n+1})\to(n,f_n)} = (n+1)\,\mu \cdot \frac{\exp\left(-\beta\left(x_{f_n}^n + \Delta - \frac{1}{\beta}\log|\mathcal{C}|\right)\right)}{\sum_{c'\in\mathcal{C}, g_{n+1}=f_n\cup\{c'\}} \exp\left(-\beta x_{g_{n+1}}^{n+1}\right)} \quad (10)$$

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 (6), but to a neighborhood around it. The detailed analysis can be found in [9].

C. Online Algorithm

We present the final algorithm that solves the dynamic **VMPR** problem in Figure 2. Upon a new job arrival, we assign the new job to one of candidate configurations according to the transition probability (8), without migrating existing jobs or modifying their routes. In a large network, there are an exponential number of neighbor states that can be reached from one state, even we allow only one VM migration. To reduce the search space, we propose a search algorithm that constructs a subgraph from an empty link set and expands it by adding links in an increasing order of marginal system costs. The algorithm $\mathbf{GreedyAdd}$ can be found [9] due to the space limitation. Upon a job departure, we pick one job from the existing tenants and probabilistically migrate it to new machines, according to (10). The candidate job to be migrated are selected among those who create the bottleneck, *i.e.*, jobs that generate flows on congested links.

VMPR-online
Upon job k arrival:
  $\mathcal{G}_k \leftarrow \mathbf{GreedyAdd}(G, f_{k-1}, k)$
  $\mathcal{F}_k \leftarrow \{f_{k-1}\} \times \mathcal{G}_k$
  Choose $f \in \mathcal{F}_k$ 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 $\mathcal{F}_j$ be the set of possible configurations by migrating j
    Calculate the transition probability of $f_j \in \mathcal{F}_j$ acc. to (10)
  end for
  $\mathcal{F}_{k-1} \leftarrow \bigcup_{j\in J} \mathcal{F}_j$
  Migrate to configuration $f \in \mathcal{F}_{k-1}$ probabilistically
  $f_{k-1} \leftarrow f$

Fig. 2. 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.

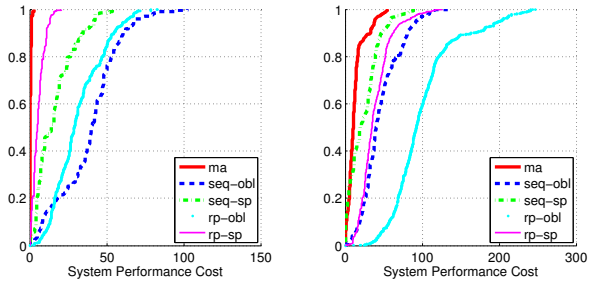
IV. Performance Evaluation

A. Evaluation Setup

To provide benchmarks for our evaluations, we consider two heuristics for server placement and two for routing selection—the combination gives four different baseline strategies.

Sequential Placement. The strategy (`seq`) places tenant VMs sequentially in the server stack, attempting to localize traffic and concentrates host subscription.

Random Placement. The strategy (`rp`) randomly picks available host machines for a job. The `rp` strategy ties to spread VMs and their traffic cross network.



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

Fig. 3. Performance of online algorithms with real work load while varying the intensity of the traffic between VMs.

Shortest-Path Routing. The strategy (`sp`) selects a path with minimum end-to-end congestion.

Oblivious Routing. The 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 online algorithm based on Markov approximation methods (`ma`) are compared to the four baseline strategies: `seq-obl`, `seq-sp`, `rp-obl` and `rp-sp`. All algorithms are evaluated on the fat-tree topology [11], which offers equal length paths between edge switches. We consider the maximum core switch utilization as the network cost, which indicates the saturation of the most congested links and is often a good estimate of the network bisection bandwidth. We run our experiment using a real workload in [12]. It contains 11 days of activity of a large cluster machines—job number, arrival time, waiting time, service time, CPU usage, memory usage, etc. More extensive evaluation results on different data center topologies and workload patternscan be found in the full version of the paper [9].

B. Evaluation of Online Algorithms

Performance under real workloads. To evaluate the performance, we extract 4,000 jobs and use its arrival times and flow durations. We vary the VM traffic intensity to characterize the system behavior under different workload patterns. Figure 3 shows the network cost of all online algorithms. In both cases, the `ma` algorithm consistently outperforms other heuristics quite significantly, because it not only takes into account the current state of jobs but also future arrivals.

Elephant vs. 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 4, 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.

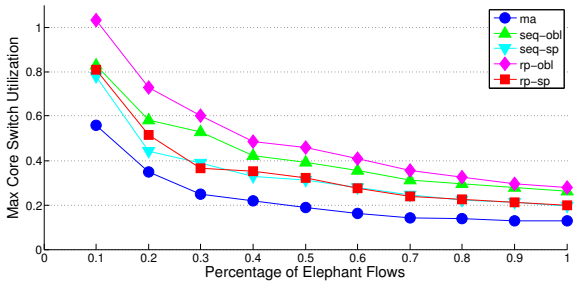


Fig. 4. Performance comparison with varying mix of elephant and mice flows.

V. Conclusion

Traditionally, VM placement and routing for data center networks are performed separately and the benefits of a joint design are unknown. We solve a joint optimization problem 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.

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] 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.
[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, M. Caesar, and J. Rexford, "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] M. Schlansker, J. Tourrilhes, Y. Turner, and J. Santos, "Killer fabrics for scalable datacenters," in *Proc. ICC*, 2010.
[8] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *IEEE INFOCOM*, pp. 519–528, 2000.
[9] 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.
[10] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," in *IEEE INFOCOM*, 2010.
[11] R. Mysore, A. Pamboris, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM*, 2009.
[12] "Logs of real parallel workloads from production systems," http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.