VirtualKnotter: Online Virtual Machine Shuffling for Congestion Resolving in Virtualized Datacenter

Xitao Wen, Kai Chen, Yan Chen
Northwestern University
Evanston, IL USA, 60208
{xwe334,kch670}@eecs.northwestern.edu
ychen@northwestern.edu

Yongqiang Liu, Yong Xia
NEC Labs China
Beijing, China, 100084
{liu_yongqiang,xia_yong}@nec.cn

Chengchen Hu
Xi'an Jiaotong University
Xi'an, China, 710049
huc@ieee.org

***Abstract*—Our measurements on production datacenter traffic together with recently-reported results [1] suggest that datacenter networks suffer from long-lived congestion caused by core network oversubscription and unbalanced workload placement. In contrast to traditional traffic engineering approaches that optimize flow routing, in this paper, we explore the opportunity to address the continuous congestion via optimizing VM placement in virtualized datacenters. To this end, we present VirtualKnotter, an efficient online VM placement algorithm to reduce congestion with controllable VM migration traffic as well as low time complexity. Our evaluation with both real and synthetic traffic patterns shows that VirtualKnotter performs close to the baseline algorithm in terms of link unitization, with only 5%-10% migration traffic of the baseline algorithm. Furthermore, VirtualKnotter decreases link congestion time by 53% for the production datacenter traffic.**

I. Introduction

Driven by technology advances and economies of scale, datacenters are becoming the mainstream hosting platform for a variety of infrastructure services (such as MapReduce [2], GFS [3] and Dryad [4]) and data-intensive applications (such as online social networking, searching, scientific computing). Today's datacenters usually form a multi-root multi-level (typically 3 tiers) tree with some oversubscription ratios at its aggregation and core layers. However, due to the massive nature of communication pattern in the datacenter network, it frequently exhibits high link utilization and even congestion at aggregation or core layers [1]. While high resource utilization is favorable for datacenter owners, network congestion can cause harmful queuing delay and packet loss, and thus affects the network throughput. These consequences could significantly degrade application performance and user experience. Therefore, addressing the congestion problem in datacenters is a meaningful goal and is the focus of this paper.

To address this problem, we resort to an increasingly adopted feature in modern datacenter - virtualization technology. Live virtual machine (VM) migration, as an important capability of virtualization technology, enables us to move a live VM from one host to another while maintaining near continuous service availability. Live VM migration provides a new dimension of flexibility - rearranging VM placement on the fly. Such spatial flexibility is proved to be effective in several scenarios, including server consolidation, power consumption saving, fault tolerance and QoS management [5]–[8]. In our case, the spatial mobility also creates an opportunity to solve the congestion problem. Through a better VM placement, we can localize a majority of traffic under ToR switches, balance the outgoing traffic, and thus resolve congestion.

However, as a limitation, live VM migration usually takes tens of seconds to transfer VM state and launch on the new host, which means a new VM placement will not take effect until all the transfers complete. Thus, to benefit from VM shuffling, we expect long-term stability in the traffic, so that we can predict the future traffic pattern and have time to adjust VM placement. Although no previous measurement directly shows the traffic stability in datacenters, the prevalence and massive nature of data-intensive applications indicate the existence of long-term traffic pattern. For example, typical applications like search engine indexing and logistic regression tend to exhibit long runtime and lasting traffic pattern as we will discuss in Section VI. Furthermore, such long-term traffic pattern is witnessed in our measurement study. As we will show in Section III, we collect and analyze an 18-hour traffic trace from a production datacenter, and observe: a) highly utilized core and aggregation network with lasting congestion pattern; and b) a well-predictable end-to-end traffic at a time granularity of tens of minutes. Such observations, coupled with the popularity of datacenter virtualization technology, point a potential avenue to address congestion via online VM shuffling.

Following this, we propose to tackle the network congestion problem through online VM shuffling. We choose to minimize the maximum link utilization, and formulate it as an optimization problem, which is shown to be a variation of the NP-hard quadratic bottleneck assignment problem (QBAP). We therefore design VirtualKnotter, an incremental heuristic algorithm that efficiently optimizes VM placement with controllable VM migration overhead. We evaluate the algorithm with various real-world and synthetic traffic patterns. We specifically compare VirtualKnotter with a clustering-based baseline algorithm that is expected to produce near-optimal link utilization. Our results suggest that VirtualKnotter achieves a link utilization performance that is close to the baseline algorithm, but with only 5% to 10% migration traffic

compared with the baseline algorithm. Our simulation further evaluates the total congestion time on each link before and after applying VirtualKnotter. The result shows VirtualKnotter is able to decrease link congestion time by 53%, demonstrating the opportunity to exploit the hourly traffic oscillation via online VM shuffling.

We summarize the main contributions of this paper as follows:

1) We collect and make an in-depth analysis on the traffic trace collected from a production datacenter[1].
2) We formulate the online VM placement problem, prove its NP-hardness, and propose VirtualKnotter.
3) We conduct extensive evaluation with both real and synthetic traffic patterns to show the optimization performance and algorithm overhead.

The rest of the paper is organized as follows. In Section II, we discuss related studies and background techniques. Next we present the measurement result in a production datacenter in Section III. Then we describe the problem formulation and the algorithm design in Section IV. In Section V, we evaluate VirtualKnotter via extensive static and dynamic simulations respectively. We discuss practical issues and limitations in Section VI before concluding in Section VII.

II. Background and Related Work

A. Traffic Engineering in Datacenter

Traffic engineering techniques have been investigated for decades. In the context of Internet, traffic engineering is usually performed by optimizing flow routing and detouring traffic away from congested links, so that the traffic is balanced and the maximal link utilization is minimized [9]–[11]. Most of those sophisticated traffic engineering techniques manipulate route via changing the link weights and coupling with link state protocols like OSPF and ISIS. While they naturally fit ISP networks with nearly random topologies and high-end routers, they may not be good options for datacenters with relatively regular topologies and commodity switches, where people usually deploy simple spanning tree forwarding and ECMP [12]. Furthermore, many recently-proposed datacenters such as BCube [13], DCell [14], PortLand [15], *etc.* have well-defined topology and the routing is largely determined by the base topology. Therefore, we cannot directly apply the existing traffic engineering techniques to these datacenter scenarios.

Recently, several traffic engineering solutions have been proposed to deal with unbalanced link utilization problem in datacenters, such as Hedera [16] and MicroTE [17]. They both proposed to arrange the traffic in flow granularity with global knowledge of the traffic load. While providing non-trivial advantages in dense structures with rich path diversity, these approaches would have marginal use when the network structure of datacenter is oversubscribed, and path diversity is limited [18]. For instance, in a traditional tree-style network, simultaneous flows have to traverse the oversubscribed core or aggregation links if the sources and destinations do not locate under the same top-of-rack switch, thus creating congestion. In this scenario, only by relocating the communication correspondents can we manage to mitigate the congestion on the core and aggregation layers. At this point, our design in this paper complements the existing approaches especially when the network is oversubscribed.

[1]The name of the production cluster is anonymized for privacy concern.

B. VM Live Migration and Application

VM live migration was first proposed and implemented by Clark, *et al.* [19], providing near continuous service during VM migration. They reported as short as few hundreds of milliseconds service downtime. Now, most of the popular VM management platforms provide support for live migration as a standard service, such as VMware vMotion [20], KVM [21], Microsoft Hyper-V Server [22], *etc.* Live migration technique delivers spatial mobility for VM placement strategy in datacenters, along with the cost of extra migration traffic, which could be 1.1x to 1.4x of VM memory footprint, or 0.34x to 0.43x if adopting appropriate compression [23]. With such extra mobility, VM placement optimization is found effective on server consolidation, power consumption saving, fault tolerance, easier QoS management and so on [5]–[8].

Recently, several studies leverage VM migration or placement to optimize the network metrics like traffic cost and end-to-end latency [24], [25]. In [24], Shrivastava *et al.* proposed to rebalance workloads across physical machines by shifting the VMs away from overloaded physical machines. The goal was to offload the overloaded physical machines while minimize the congestion caused by the migration traffic. In [25], Meng *et al.* proposed to minimize the traffic cost, which is quantified in terms of traffic volume times the communication distance, via VM placement. They proposed a min-cut clustering-based heuristic algorithm whose runtime complexity is $O(n^4)$, where n is the number of VMs. Worse, their algorithm did not take into account the VM migration traffic, leading to a near complete shuffling of almost all VMs in each round. Relative to these works, VirtualKnotter minimize the continuous congestion mainly in core and aggregation links with runtime overhead of $O(n^2 \log n)$ and controllable migration traffic, which enables online VM replacement at the granularity of tens of minutes.

III. Measurement and Motivation

Recent measurement results in datacenter illustrate several remarkable traffic properties in datacenter network.

- Congestion is ubiquitous in datacenter network. Specifically, it is reported not rare to see above 70% link utilization at a timescale of 100 seconds [1]. Such a high utilization can cause serious packet drop, significant queuing delay at the congested spots, and thus impacts overall throughput. Those effects can degrade application performance with both large data transfer and small request-response flows.
- Datacenter network is frequently the bottleneck to application-layer performance. For instance, Chowdhury *et al.* show communication time account for 42%-70%

running time in MapReduce services [26]. This means a decrease of 10% in communication time will result in 4.2%-7% performance gain, which is hard to achieve by speeding up computing.

- Link utilization is highly divergent among core and aggregation links within a datacenter. It is shown that the highly utilized hot links usually account for less than 10% of all the core and aggregation links, while other links remain lightly utilized with utilization less than 1% [18], [26]. Such phenomenon indicates the spatially unbalanced utilization of network resources may be one of the causes of the network congestion. This implies keeping a spatially balanced resource demand may potentially have the same benefit as provisioning bandwidth capacity at hot spots.

However, despite those observations from previous measurement studies, some traffic properties like congestion pattern and long-term traffic stability still remain unclear. During our study, we learn that those properties are essentially helpful to design a traffic engineering scheme as well as to determine the parameters for a specific datacenter.

In this section, we focus on obtaining quantitative knowledge of congestion pattern (where and how long does congestion occur?) and traffic stability at various granularities (how stable is the traffic, viewing in the time scale of seconds, minutes or hours?). We collected traffic matrices from a production cluster with 395 servers, which run MapReduce-like services. This cluster has a hierarchical structure with 4:1 oversubscription ratio. We aggregate the traffic matrices for every 30 seconds, as shown in Figure 1. The dataset lasts consecutively for about 18 hours.

Our measurement reveals three key observations with implications for the VirtualKnotter design. First, we find a majority of congestion events last for tens of minutes, while the set of congested links evolves over time. This observation demonstrates the long-term communication pattern of the upper-layer application, implying the potential benefit to conduct traffic engineering at a timescale of tens of minutes. Second, congestion events tend to be local, usually involving less than 30% of links, which indicates temporarily unbalanced traffic in the datacenter. Finally, we observe that over 60% traffic volume is relatively stable at an hourly granularity. This property allows for the prediction of future traffic matrix with the previous ones, which is the key assumption of traffic engineering techniques.

A. Congestion within Datacenter

a) Traffic Concentration: Figure 1 shows a typical traffic matrix in our dataset. It presents a busy traffic matrix with severe local congestion, which involves six out of ten racks within the cluster. We can see the traffic is highly concentrated within and across the upper and lower part of the cluster. In fact, with further inspection into the link utilization, we find the links among those racks have an average utilization of 65% in the core and aggregation layer, with the highest of 80.3%. In the meantime, links associated with middle four racks remain relatively idle.

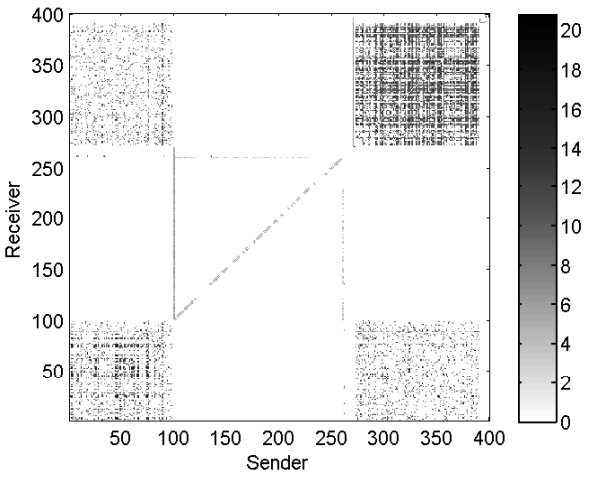


Fig. 1. An observed 30-sec traffic matrix. Each data point is the traffic volume from a sender (x axis) to a receiver (y axis). Gray scale reflects traffic volume in natural log of bytes.

b) Location and Duration of Congestion Events: To further understand the spatial distribution and temporal duration of congested links, we locate the congested links by plotting them into a time series figure, as shown in Figure 2. We pick 60% utilization as the congestion threshold, but other thresholds like 65% or 70% yields qualitatively similar results. We define the term *congestion event* as the period of time when the set of congested links keeps the same without discontinuity of longer than five minutes. Using this definition, we examine the congestion events in our trace, resulting in two interesting findings. First, congestion events exist and tend to be local during the observation period, with no congestion event involving more than half of links. Instead, a typical congestion event just involves about 1/3 of core and aggregation links. Moreover, different congestion events may consist of quite different sets of congested links. This phenomenon indicates that the application's communication demand can distribute highly unevenly within a datacenter, and that the traffic distribution evolves over time. Second, a congestion event tends to last for an extended period of time. We totally observe seven congestion events that last for at least 20 minutes long. We further speculate such a continuous congestion event may indicate an application-layer transfer event, such as a MapReduce shuffle between mappers and reducers.

B. Traffic Stability Analysis

Although there are measurement results demonstrating poor predictability of traffic matrix at the timescale of tens of milliseconds to hundreds of milliseconds [18], [27], we still have little knowledge about long-term traffic stability in datacenters. In this subsection, we design a stableness indicator and conduct measurement in the dataset. Formally, the stableness

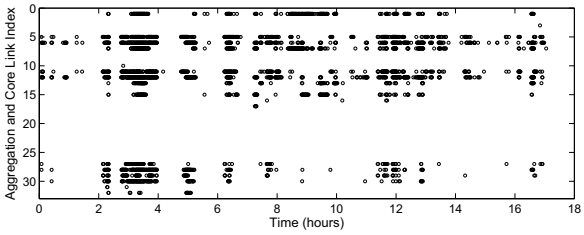


Fig. 2. Time and locations of congestion observed in the datacenter. Each circle represents a 30-second congestion occurred on a core or aggregation link.

indicator is defined by

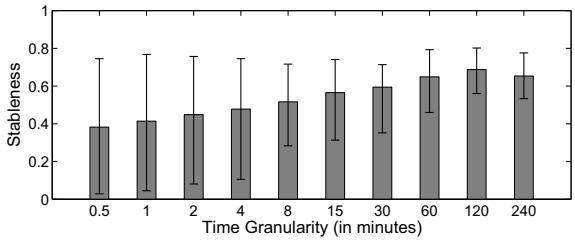
$$Stableness(t_{prev}, t_{curr}) = \frac{min(t_{prev}, t_{curr})}{max(t_{prev}, t_{curr})}, \quad (1)$$

where t_{prev} and t_{curr} stand for traffic volume in the previous epoch and the current epoch respectively. The fundamental idea for stableness indicator is to estimate percentage the stable part comparing two consecutive traffic states. For a traffic matrix, we calculate a single stableness value for each element. Then, we select 10% percentile, median, and 90% percentile as the indicator of the entire distribution, as shown in Figure 3(a). Similar procedure is used to generate Figure 3(b).

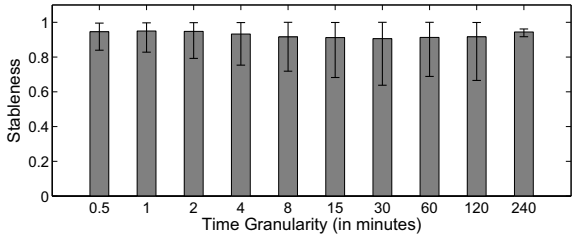
We can simply interpret the stableness indicator as the percentage of stable traffic volume in two consecutive epochs. Figure 3(a) illustrates the stableness of end-to-end pairwise traffic varying the timescales from 30 seconds to 4 hours. We can see a range of 40% to 70% of traffic volume can be expected stable, peaking at the timescale of 2 hours, which demonstrates a good hourly predictability of the traffic matrix in our dataset. Although the traffic stability varies greatly at small timescales, the hourly traffic factor tends to concentrate on about 60% with small oscillation. Furthermore, Figure 3(b) demonstrates even better stability on core and aggregation links, with the median stable traffic indicator larger than 90%. The better stableness of upper-layer links is a result of traffic aggregation as well as the constantly higher utilization.

Our experiment results reveal highly stable traffic demand at an hourly granularity in a datacenter. Actually, such high stable trace is not obtained by chance. Data-intensive applications tend to exhibit a similar stability, due to the massive nature of data transfer and long computing time on distributed computing node. We will discuss the traffic stability issue later in Section VI.

The above findings motivate us towards an online VM placement approach for congestion resolving, as the existing of continuous congestion events, evolving congestion patterns and good traffic predictability. We argue a considerable part of continuous link congestion can be eliminated, or at least mitigated, by evenly distributing outgoing traffic and localizing intra-datacenter traffic within a rack or nearby racks. By exploiting the spatial mobility of VM placement in a datacenter, we can potentially achieve both localized and balanced communication pattern, and thus benefiting from higher throughput and lower queuing latency.



(a) Stableness indicator of pairwise traffic volume



(b) Stableness indicator of core and aggregation links

Fig. 3. The fraction of traffic volume remains stable in two successive periods. The bar value shows the median and error bar shows 10% and 90% percentile.

IV. Design

In this section, we first formulate the online VM placement problem using integer optimization language and analyze its complexity. Then, we propose VirtualKnotter, a two-step heuristic algorithm for efficient online VM placement.

A. Online VM Placement Problem

According to previous discussion, we want to achieve following goals in the online VM placement scheme:

1) The optimized VM placement should minimize the congestion status measured by a link congestion objective, such as maximum link utilization.
2) The migration traffic should be controllable, i.e., parameters should be provided to control the number of migrated VMs between the current VM placement and the optimized VM placement.
3) The algorithm should be scalable, i.e., the runtime overhead should be considerably less than the target replacement timescale (tens of minutes) for a typical sized datacenter.

Given above principles, we formulate the online VM placement problem as follows.

Assumptions. We assume the datacenter is connected with a hierarchical structure, such as a tree or multi-root tree. Note that we aim to address congestion problem, which theoretically does not exist in non-blocking network, such as fat tree or VL2. Thus, we exclude those network structures from the scope of our study. A server's ability to host VM is constrained by the server's physical capacity, such as CPU/memory. Thus, we assume a known number of VM h_s can be hosted on a

certain server s, referring as VM slots. We further assume a deterministic single-path routing in the datacenter network.

Input & Output. The online VM placement problem accepts network routing P, traffic matrix M, external traffic E and current VM placement X' as input, and generates optimized VM placement X as output. We denote the network routing by a binary-value function $P_{s,d}(l)$, meaning whether the traffic path from server s to d traverses through link l. Similar notation $P_s(l)$ represents the routing path going outside the datacenter, meaning whether the traffic path from server s to the gateway traverses through link l. As the datacenter runs, we assume the traffic matrix $M_{i,j}$ and external traffic E_i for a certain period of time are also available. $M_{i,j}$ denotes the traffic volume from VM i to VM j. E_i denotes external traffic volume from VM i to the gateway. Note, such traffic statistics can be collected either by ToR switches or VM hypervisors on each server without incurring considerable overhead. Moreover, the problem also takes the current VM placement matrix X' as input for incremental VM placement. The output is the optimized VM placement matrix X. Both $X_{i,s}$ and $X'_{i,s}$ are binary-value matrix indicating whether VM i is placed on server s.

Objective. We choose the maximum link utilization (MLU) as the optimization objective. The MLU is determined by the highest utilized link, which characterizes the worst congestion status in a network during a period of time. Given the MLU is widely adopted as the optimization goal in the context of Internet traffic engineering [9]–[11], we believe it will also be effective to represent the overall congestion status in a datacenter network. With the preceding notations, we formally define the following objective function

$$\min_{X} \max_{l} \frac{T(l,X)}{C(l)}, \tag{2}$$

$$T(l,X) = \sum_{s,d} P_{s,d}(l) \sum_{i,j} X^T_{i,s} M_{i,j} X_{j,d} + \sum_{s} P_s(l) E_i X_{i,s},$$

where s (or s,d pair) enumerates all hosts (or host pairs) and i,j pair enumerate all VM pairs. X is a valid VM placement matrix, which satisfies two constraints: $\sum_j X_{i,j} = 1$ and $\sum_i X_{i,j} \leq h_j$. And l is a physical link between two switches. Another constraint concerning the VM migration traffic is modeled by the number of VMs needed to be migrated. The migrated VM number is limited to be no greater than an input threshold Th: $\frac{1}{2} sum|X_{i,s} - X'_{i,s}| \leq Th$.

In the objective function, two parts in $T(l,X)$ represent respectively the internal traffic and external traffic traversing a given link l. The inner maximum operator enumerates all links, so as to seek for the MLU. The outer minimum operator finds the lowest MLU among all valid VM placements. Therefore, this objective function is to minimize the MLU.

Complexity. The above optimization problem falls into the category of Quadratic Bottleneck Assignment Problem (QBAP), which is a known NP-hard problem [28]. It is conceivable that the variables $X_{i,s}$ have quadratic form in the objective function and the overall problem is an assignment problem with a quadratic bottleneck goal (minimum of maximum). We formally give the complexity proof by reducing the QBAP problem to our problem in the Appendix.

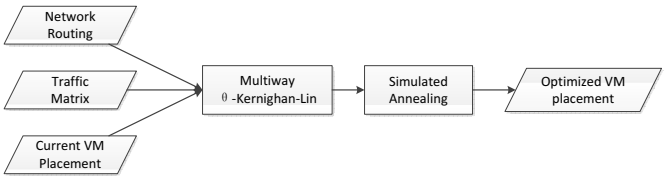


Fig. 4. Algorithm flowchart of VirtualKnotter. The diamond boxes are input/output data, and the rectangle boxes are functions.

B. Algorithm

In this subsection, we propose VirtualKnotter, a heuristic algorithm to the online VM placement problem. We have shown our problem is inherently NP-hard; and no efficient exact solution can scale to the size of a typical datacenter. Therefore, we resort to an intuitive heuristic approach.

Intuition 1: **Incremental local search is preferred rather than clustering, in order to satisfy migration traffic constraint.** So far as our knowledge goes, there is no clustering algorithm that is able to balance the optimality and number of elements that move across clusters. However, for local search, it is inherently easy to keep track of the searching depth, which is equivalent to migration traffic in our case.

Intuition 2: **A good initial placement is needed to speed up local search.** Searching usually takes long and uncertain amount of time, which we do not want in online algorithm. A possible way to speed up searching is to provide a good estimation as the initial solution. We find that an efficient algorithm that improves traffic localization may satisfy our requirements. More localized traffic means that greater percentage of data is exchanged within a rack or nearby racks. Although not exactly the same, traffic localization shares similar objective with the MLU goal. This is because congestion often occurs at core and aggregation links, where the localization algorithm aims to offload traffic from.

Based on the above intuitions, we propose a two-step heuristic algorithm: we borrow and adapt the multi-way Kernighan-Lin graph partitioning algorithm to generate initial VM placement with better localized traffic [29], [30]. Then, we employ the simulated annealing algorithm to further optimize the MLU. To avoid excessive VM migration, we only invoke VirtualKnotter when the minimum invocation time interval is met and link congestion is observed in the network. Figure 4 shows the high-level logic flow of the algorithm. And we present the pseudo-code in Algorithm 1, Algorithm 2 and Algorithm 3.

1) Multiway θ-Kernighan-Lin Algorithm: The key idea of the Kernighan-Lin graph partitioning algorithm is to greedily swap elements across clusters, thereby iteratively reduce the overall weight of a graph cut. We adapt the algorithm by introducing a *migration coefficient* θ, in order to constrain

Algorithm 1 Multiway θ-Kernighan-Lin Procedure

Require: M(Traffic matrix), T(Network topology), X(Current VM placement)
  for all $layer$ in T **do**
    Sort element set $E \in X$ on $layer$ by outgoing traffic
    while $|E| \geq 2$ **do**
      Split elements into two half interleavingly, resulting in S_1 and S_2
      θ-Kernighan-Lin-Improve(M, S_1, S_2)
      $E \leftarrow S_1$ and S_2 respectively
    end while
  end for
  return X

Algorithm 2 θ-Kernighan-Lin-Improve

Require: M(Traffic matrix), S_1, S_2(VM sets), θ(Migration coefficient)
  $CurrGain \leftarrow 0, Gain \leftarrow$ {empty list}
  Initialize the migration gain $D(i)$,
  where $D(i) = \sum_{j \notin S(i)} M(i,j) - \sum_{j \in S(i)} M(i,j)$
  for $s = 1$ to $\frac{1}{2}\theta * min(len(S_1), len(S_2))$ **do**
    Swap the VM pair $(i,j) \in (S_1, S_2)$, which has maximum $G(i,j) = D(i) + D(j) - 2 * M[i,j]$
    $CurrGain = CurrGain + G(i,j)$
    $Gain.append(CurrGain)$
    Update D: $D(k) = D(k) + M(k,j) - M(k,i)$, if $k \in S_1$
      $D(k) = D(k) + M(k,i) - M(k,j)$, if $k \in S_2$
  end for
  return $max(Gain)$ and corresponding VM sets S_1', S_2'

Algorithm 3 Simulated Annealing Procedure

Require: M(Traffic matrix), P(Network routing), X'(Current VM placement), N_{max}(Max iterations), θ(Migration coefficient)
  $X, X_{best} \leftarrow X'$
  $E, E_{best} \leftarrow Energy(M, P, X)$
  for $T \leftarrow N_{max}$ to 0 **do**
    $X_{new} \leftarrow Neighbor(X)$
    $E_{new} \leftarrow UpdateEnergy(M, P, X)$
    if $P(E, E_{new}, T) > Rand()$ **and** $Diff(X, X') < \theta$ **then**
      $X \leftarrow X_{new}$, $E \leftarrow E_{new}$
    end if
    if $E < E_{best}$ **then**
      $X_{best} \leftarrow X$, $E_{best} \leftarrow E$
    end if
  end for
  return X_{best}

the migration cost of improved VM placement. This heuristic algorithm is originally used in the layout design of circuits and components in VLSI [31], where an efficient heuristic solution for the minimum graph cut problem is needed. In our scenario, we adapt the Kernighan-Lin algorithm for the purpose of improving the traffic localization and reduce the traffic load on core and aggregation layers. The algorithm runs on the original VM placement hierarchically in a top-down manner. In each layer, it bisects the VM clusters and calls θ-Kernighan-Lin-Improve procedure for bisection improvement. The procedure swaps elements between two clusters iteratively and greedily according to the *Gain* on the cut weight reduction. Note that the number of iterations is limited by migration coefficient θ, so as to avoid VM swaps which only bring marginal benefit. The runtime complexity of Multiway θ-Kernighan-Lin Algorithm is $O(n^2 \log n)$, where n is the number of VMs.

2) Simulated Annealing Searching: In this step, we need to efficiently search for a fine-grain solution of minimizing MLU. We employ the simulated annealing algorithm, which is known efficient in searching in an immense solution space. The initial VM placement accepts as input the output of the multiway θ-Kernighan-Lin algorithm. The function $Energy$ estimates and returns the MLU for a given VM placement. In each iteration, a neighboring state $Neighbor(X)$ is generated by swapping a VM pair that can offload traffic from the most congested link. To find such a VM pair, we conceive a heuristic: we seek for two distinct and heavily communicated pairs over the congested link, pick one VM from each pair, and swap them. Then, we move to a neighboring state with a certain acceptance probability P, which depends on the energy of current and neighboring placement as well as current temperature T. The acceptance probability we use is defined as

$$P(E, E_{new}, T) = \begin{cases} 1 & \text{if } E_{new} < E \\ e^{c(E - E_{new})/T} & \text{if } E_{new} \geq E \end{cases}$$

The temperature is decreased with each iteration until stopped at zero, allowing a higher probability to move to a worse placement with a high temperature. This behavior allows simulated annealing algorithm to avoid stuck at the local minima. The complexity of the simulated annealing has two components: the initialization requires $O(n^2)$; each simulated annealing iteration requires $O(n)$. Thus, the overall complexity is $O(n^2 + N_{max} * n)$, where N_{max} is maximum number of iterations.

V. Evaluation

In this section, we describe our evaluation of VirtualKnotter in three aspects: static performance, overhead and dynamic performance. The goal of these experiments is to determine the benefit as well as the cost when deploying VirtualKnotter and the baseline algorithms.

A. Methodology

1) Baseline Algorithms: We compare VirtualKnotter with a clustering-based placement algorithm. The advantage of clustering algorithm lies in the fact that clustering produces near optimal traffic localization. However, to the best of our knowledge, clustering algorithm cannot be trivially adapted to perform an incremental optimization, which implies nearly 100% of VMs need to be migrated in each round. Also,

clustering algorithms usually have a runtime complexity no less than $O(n^3)$, where n is the number of VM number. Thus, in our evaluation, we treat the clustering algorithms as a reference of the optimization performance without limit on runtime and migration traffic. Among many available clustering algorithms, we select a variation of hierarchical clustering algorithm, which is able to specify the cluster size constraint [32]. We run the clustering algorithm following a top-down order according to the network topology. We later map each cluster into a switch and VM into a physical machine. The runtime complexity of the algorithm is $O(n^3)$. The detailed description is referred to the original paper [32].

We also select each single step of VirtualKnotter, namely the Multiway θ-Kernighan-Lin algorithm (KL) and the simulated annealing algorithm (SA) as baseline algorithms. The purpose is to show how the combination of algorithms actually benefits compared with individual steps.

2) Communication Suites:

- **Real-world Traces**: We use the collected traffic trace described in Section III. The traffic trace is collected from a production cluster with 395 servers and an oversubscription ratio of 4:1 in the core layer. The collected data has a time granularity of 30 seconds, and lasts for nearly 18 hours.
- **Measurement-based Patterns**: In order to test the scalability of the algorithm, we derive the measurement-based patterns from the measurement results by Kandula *et al.* [1]. First, we derive host communication pattern from both the inter-rack and intra-rack correspondent distribution. Then, we assign traffic volume to each pair of hosts, according to the traffic volume distribution. The VM number is 10K, and the physical topology is assumed hierarchical with an oversubscription ratio of 10:1.
- **Hotspot Patterns**: Recent measurement revealed highly skewed traffic patterns often exist in production datacenter, known as hotspot pattern [18] [33]. We synthesize such traffic pattern by randomly select ToR switches as hotspots, connect the individual servers under hotspots with a number of servers under normal ToRs, and assign a constant large traffic volume to each connection. The VM number is 10K, and the physical topology is assumed hierarchical with an oversubscription ratio of 10:1.

3) Metrics and Settings: First, we evaluate the static algorithm performance by measuring the maximum link utilization. We compare VirtualKnotter against KL, SA and clustering algorithm, as well as the original VM placement without any optimization. Then, we evaluate the algorithm overhead in the sense of both additional migration traffic and algorithm runtime. Finally, we simulate the real scenario and evaluate the overall algorithm performance on dynamic congestion resolving. We replay the time-series traffic and run the algorithm on current traffic pattern, resulting in an optimized VM placement. Then, we apply the optimized VM placement on the next traffic pattern, and inject additional VM migration traffic into network. We compare the link congestion time (link$*$minute) varying the replacement timescale. Note, the migration coefficient θ is set to 0.1 in both KL and SA.

B. Static Performance

Figure 5 shows the maximum link utilization before and after applying the algorithms. Every data point represents a communication pattern of a collected or synthetic traffic matrix. We try to minimize the maximum link utilization; thus the curve close to the upper left corner is favorable. From the figures, we find VirtualKnotter significantly outperforms both KL and SA, and has a similar static performance as Clustering algorithm, which serves as a reference to the upper bound. This result illustrates that through combination VirtualKnotter provides qualitative improvement over both KL and SA. It is worth to note that VirtualKnotter requires significantly less VM migration compared with Clustering-based algorithm (5%-10% vs. $\sim 100\%$), which will be analyzed in detail in the following subsection.

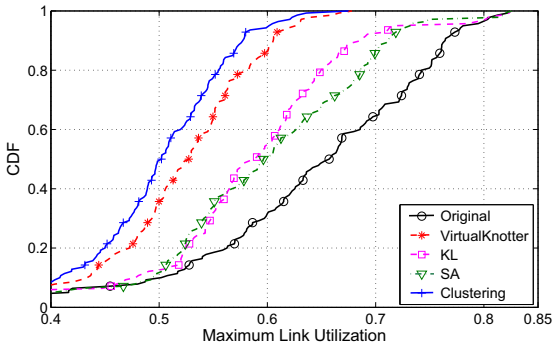
C. Overhead

1) Migration Traffic: VM migration introduces considerable bulk data transfer into the datacenter network. To understand the counter-effect, we need to quantitatively measure how large volume of the additional migration traffic we should expect for each algorithm. We model the VM migration as bulk data transfer. We assume each VM has a memory footprint of 2 gigabytes, which will result in 2.2 to 2.8 gigabytes bulk transfer using live migration [34]. Thus, we take the median 2.5 gigabytes as the extra traffic volume for each migrated VM in the simulation. We plot the relative traffic volume for both VirtualKnotter and the baseline algorithm in Figure 6. From the figure, we observe that the baseline algorithm introduces around 10% traffic volume of goodput with the timescale of 30 minutes. The traffic overhead of VirtualKnotter is over one order of magnitude less than Clustering algorithm, which ranges between 0.2% and 1% of goodput.

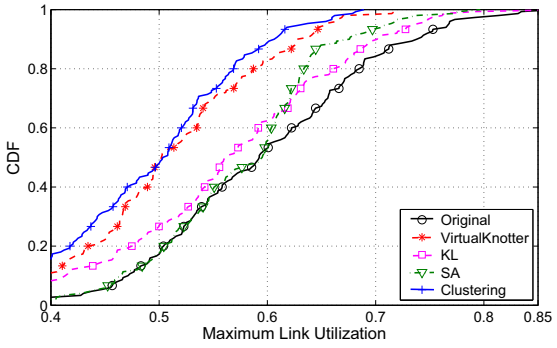
2) Algorithm Runtime: The runtime overhead of VirtualKnotter is shown in Figure 7. It is evident from the figure, that VirtualKnotter consumes tens of seconds for a typical virtual datacenter or tenant with thousands of VMs, and scales much better than the baseline algorithm. Also the runtime overhead of VitualKnotter is considerably less than the target replacement granularity which is tens of minutes, thereby enabling the online VM replacement.

D. Dynamic Simulation

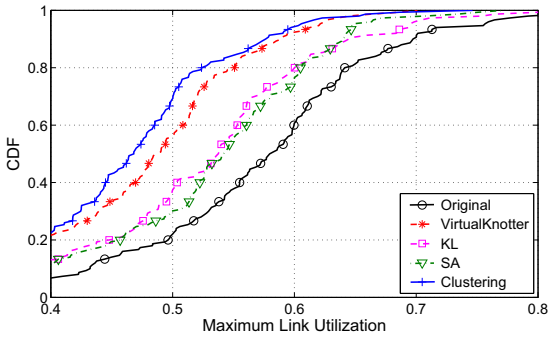
We conduct simulation to evaluate the dynamic performance of both VirtualKnotter and the baseline algorithm considering migration traffic. We replay the real-world traces and run both algorithms at a variety of timescales. The optimized VM placement resulted from previous period of time is deployed on the next period, introducing sudden migration traffic burst. The migration traffic is modeled exactly the same as in Subsection V-C. Figure 8 shows the total link congestion time (link$*$time) of all core and aggregation links varying replacement granularity. The figure demonstrates that, even



(a) CDF of MLU on Real-world Traces (VM#= 395)



(b) CDF of MLU on Measurement-based Patterns (VM#=10K)



(c) CDF of MLU on Hotspot Patterns (VM#=10K)

Fig. 5. Static algorithm performance in terms of maximum link utilization.

considering migration overhead, VirtualKnotter still manages to harvest the benefit of over one half less link congestion time at timescales of 30 minutes, one hour or two hours. On the contrary, the baseline algorithm, due to the migration traffic, exhibits a worse congestion status than original placement, with a 7.8% to 230.6% higher link congestion time compared with original placement.

Figure 9 presents the locations and durations of congestion before and after applying VirtualKnotter. From the figure, we can see that most of the continuous congestion events are resolved by VM replacement right after detected, demonstrating VirtualKnotter is effective on resolving long-lived congestion

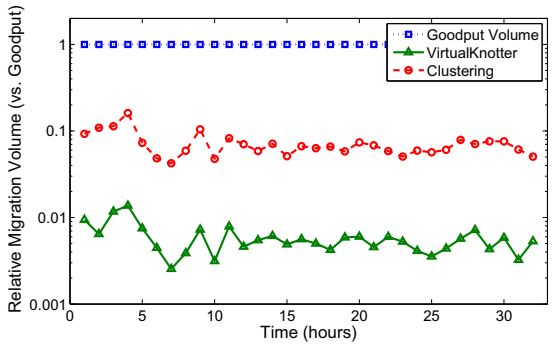


Fig. 6. Relative migration traffic compared with network goodput. The timescale is 30 minutes.

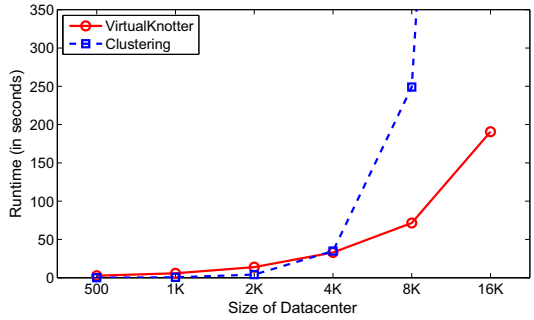


Fig. 7. Algorithm runtime overhead varying the size of a datacenter.

event. The benefit is harvested at the cost of dispersed short-lived congestion events caused by the burst of migration traffic. Such congestion events are normally aligned with VM replacement events, and last for only one to two minutes. We believe such ephemeral congestion events are more tolerable by applications in datacenter and yield far less negative effects compared with the continuous congestion events.

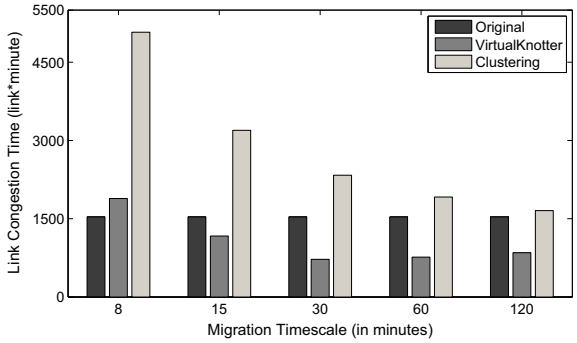
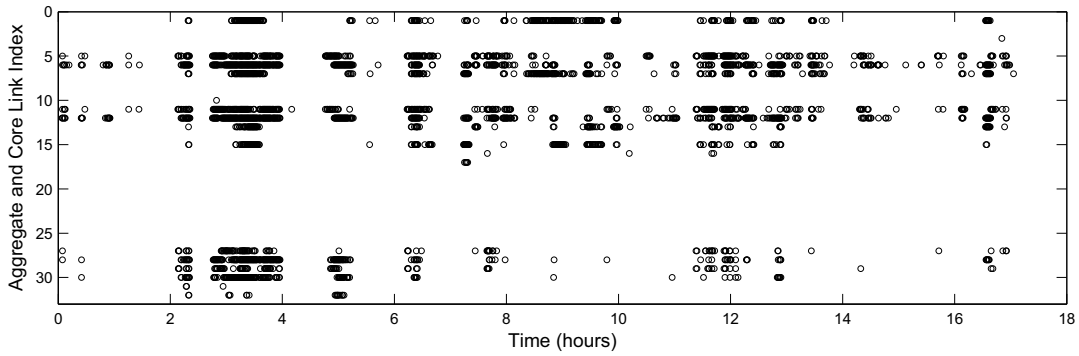
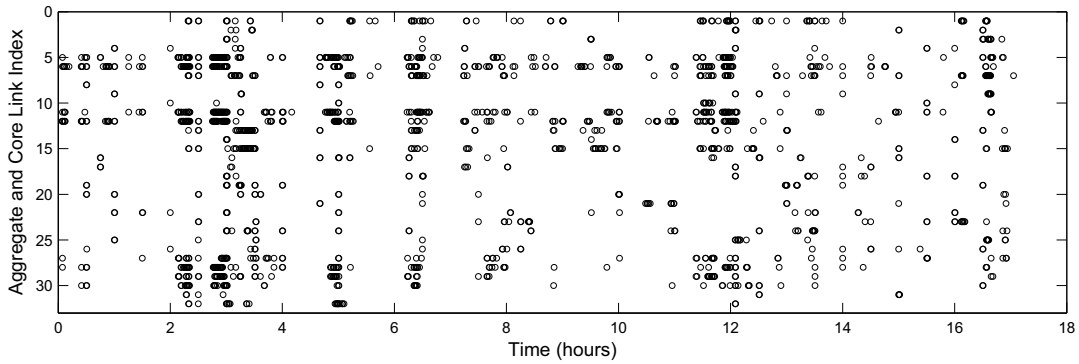


Fig. 8. Link congestion time varying replacement timescale. Congestion threshold is 0.6. All the congested time of core and aggregation links are included.



(a) Congestion timeseries before VirtualKnotter



(b) Congestion timeseries after VirtualKnotter

Fig. 9. Algorithm performance in term of congestion timeseries. Each circle represents a 30-second congestion occurred on a core or aggregation link.

VI. Discussion

Traffic Stability: VirtualKnotter manifests good performance on resolving static congestion with VM placement shuffling. However, in a real-world setting, we have to predict the future traffic pattern based on history records; thereby the traffic stability may have considerable impact on the accuracy of prediction. We admit that VirtualKnotter is not suitable for datacenters with highly dynamic traffic. Although we observe an average proportion of 40% to 70% of traffic remains stable within our target timescales, we understand the traffic properties highly depend on the application layer. We argue that data-intensive applications would most likely exhibit a similar stability with our measurement result due to the massive nature of data pre-fetch, intermediate result shuffle and result transfer. For example, the distributed indexing time for a search engine is estimated over 10000 seconds and 4000 seconds in the map phase and the reduce phase respectively by McCreadie *et al.* [35]. During the whole period, network can suffer from high utilization with a continuous traffic pattern. Logistic regression, as a representative machine learning algorithm, involves tens of MapReduce iterations, which lasts for an hour to train the model on a 29GB dataset [36]. Again, traffic pattern among iterations is not likely to change greatly. Therefore, we believe there are a part of data-intensive applications that will exhibit a similar long-term stability in datacenters.

One-time placement scheduling vs. Dynamic replacement: We propose a dynamic VM replacement scheme in this paper. For those datacenter whose traffic pattern evolves over time, we have shown that dynamic replacement scheme can effectively harvest from the traffic oscillation. However, we notice that there exist applications with relatively constant traffic patterns. For those applications, it is probably sufficient to conduct one-time placement scheduling rather than dynamic replacement. We leave the identification of such constant traffic pattern and determination of best replacement timescale as our future work.

VII. Conclusion

In this paper, we present VirtualKnotter, a novel online VM placement algorithm for resolving link congestion in datacenter network. VirtualKnotter exploits the flexibility of VM placement in virtualized datacenter, for the purpose of improve traffic localization and balance link utilization. VirtualKnotter strikes a good balance between resolving congestion and introducing migration traffic, resulting in an efficient and

practical VM placement algorithm. We evaluate VirtualKnotter via static experiments and extensive dynamic simulation. Our results suggest that VirtualKnotter delivers over 50% of congested time reduction with negligible overhead.

ACKNOWLEDGMENT

We thank Kenny Tay for his work on the initial problem identification and exploration. Part of the work was done when Xitao Wen and Kai Chen were summer interns at the NEC Labs China.

REFERENCES

[1] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *IMC '09*.
[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, pp. 107–113, January 2008.
[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *SOSP '03*.
[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, pp. 59–72, March 2007.
[5] R. Nathuji and K. Schwan, "Virtualpower: coordinated power management in virtualized enterprise systems," in *SOSP '07*.
[6] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," in *Integrated Network Management 2007*.
[7] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for hpc with xen virtualization," in *ICS '07*.
[8] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "Enacloud: An energy-saving application live placement approach for cloud computing environments," in *CLOUD '09*.
[9] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "Cope: traffic engineering in dynamic networks," in *SIGCOMM '06*.
[10] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, *Overview and Principles of Internet Traffic Engineering*. RFC Editor, 2002.
[11] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs," in *SIGCOMM '03*.
[12] A. Dixit, P. Prakash, and R. Rao Kompella, "On the efficacy of fine-grained traffic splitting protocols in data center networks," in *SIGCOMM '11*.
[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *SIGCOMM*, 2009.
[14] 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 *SIGCOMM*, 2008.
[15] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *SIGCOMM*, 2009.
[16] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," *NSDI '10*.
[17] T. Benson, A. Anand, A. Akella, and M. Zhang, "The case for fine-grained traffic engineering in data centers," in *INM/WREN '10*.
[18] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *IMC '10*.
[19] 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 '05*.
[20] VMware. (2011, Nov.) Vmware vmotion for live migration of virtual machines. [Online]. Available: http://www.vmware.com/products/vmotion/overview.html
[21] KVM. (2011, Nov.) Kvm migration. [Online]. Available: http://www.linux-kvm.org/page/Migration
[22] Microsoft. (2011, Nov.) Hyper-v live migration. [Online]. Available: http://technet.microsoft.com/en-us/library/dd446679(WS.10).aspx
[23] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression," *2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–10, 2009.
[24] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *INFOCOM '11*.
[25] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM '10*.
[26] M. Chowdhury, M. Zaharia, J. Ma, M. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *SIGCOMM '11*.
[27] A. Greenberg, 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.
[28] R. E. and Burkard, "Selected topics on assignment problems," *Discrete Applied Mathematics*, vol. 123, no. 1-3, pp. 257 – 302, 2002.
[29] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical J.*, vol. 49, p. 291, 1970.
[30] M. W. Sung, S. Kyu, L. Jason, and C. M. Sarrafzadeh, "Multi-way partitioning using bi-partition heuristics," in *ASPDAC 2000*.
[31] C. P. Ravikumar, *Parallel Methods for VLSI Layout Design*. Westport, CT, USA: Greenwood Publishing Group Inc., 1995.
[32] T. Asano, P. Bose, P. Carmi, A. Maheshwari, C. Shu, M. Smid, and S. Wuhrer, "A linear-space algorithm for distance preserving graph embedding," *Comput. Geom. Theory Appl.*, vol. 42, May 2009.
[33] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting data center networks with multi-gigabit wireless links," in *SIGCOMM '11*.
[34] 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 '05*.
[35] R. Mccreadie, C. Mcdonald, and I. Ounis, "Comparing Distributed Indexing: To MapReduce or Not?" in *7th Workshop on Large-Scale Distributed Systems for Information Retrieval*, 2009.
[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *HotCloud*, 2010.

APPENDIX
PROOF OF PROBLEM COMPLEXITY

We prove the complexity of the online VM placement problem (OVMPP) by reducing quadratic bottleneck assignment problem (QBAP) to OVMPP [28]. The QBAP problem is formally described as following: given three n-by-n matrices $A = (a_{ij})$, $B = (b_{kl})$ and $C = (c_{ik})$ with non-negative real values. One wants to find the optimal permutation $\phi(i)$, which satisfies a one-to-one mapping from N to N ($N = \{1, 2, ..., n\}$). The objective function is

$$\min_{\phi} \max_{1 \le i,j \le n} a_{ij} b_{\phi(i)\phi(j)} + c_{i\phi(i)}.$$

Now, we want to map objective Function 2 to the above goal, to show for any QBAP instance we can transform it to an OVMPP in polynomial time. In fact, we can rewrite assignment matrix X to permutation $\phi(i)$ by assign different index for each VM slot. We further assume a full-meshed network, where each source destination pair (s, d) uniquely connected by a physical link $l(s, d)$. Thus, we have the following reduction mapping

$$\begin{aligned}
a_{ij} &= \sum_{i,j} P_{i,j}(l(i,j)) \\
&= P_{i,j}(l(i,j)), \text{(since (s,d) maps to a unique link)} \\
b_{\phi(i)\phi(j)} &= \sum_{\phi(i),\phi(j)} X^T_{\phi(i),i} M_{\phi(i),\phi(j)} X_{\phi(j),j} \\
c_{i\phi(i)} &= \sum_{i} P_i(l(i)) E_{\phi}(i) X_{\phi(i),i}.
\end{aligned}$$

The above mapping enables us to transform any QBAP to an OVMPP, implying QBAP is no harder than OVMPP. Since QBAP is known NP-hard, we conclude the OVMPP is also NP-hard.