PAM & PAL: Dynamic Virtual Machine Placement and Migration in Policy-Aware Data Centers with Heterogeneous Resource Demands

Christopher Gonzalez, Hugo Flores, Vincent Tran, and Bin Tang
Department of Computer Science
California State University Dominguez Hills, Carson, CA 90747, USA
Email: {cgonzalez393,hflores27,vtran42}@toromail.csudh.edu, btang@csudh.edu

***Abstract*—We focus on policy-aware data centers (PADCs), wherein virtual machine (VM) traffic traverses a sequence of middleboxes (MBs) for security and performance purposes, and propose two new VM placement and migration problems. We first study PAL: policy-aware virtual machine placement. Given a PADC with a data center policy that requires communicating VM pairs to satisfy, the goal of PAL is to place the VMs within the PADC to minimize their total communication cost. Due to dynamic traffic rates in PADCs, however, the above VM placement may no longer be optimal after some time. We thus study PAM: policy-aware virtual machine migration. Given an existing VM placement in the PADC and dynamic traffic rates among communicating VMs, PAM migrates VMs to minimize the total cost of migration and communication for the VM pairs. We consider both uniform and heterogeneous resource demands of VMs, and design optimal, approximation, and heuristic *policy-aware* VM placement and migration algorithms. Our experiments show that i) VM migration is an effective technique to mitigate dynamic traffic in PADCs, reducing the total communication cost of VM pairs by 25%, ii) our PAL algorithms outperform a state-of-the-art VM placement algorithm that is oblivious to data center policies by 40-50%, and iii) our PAM algorithms outperform the existing policy-aware VM migration scheme by 30%.**

***Index Terms*—Policy-Aware Data Centers, Virtual Machine Placement, Virtual Machine Migration, Algorithms**

I. Introduction

Background. Cloud data centers are the dominant computing infrastructure for the IT industry as well as an integral part of our Internet fabric. Enabled by virtualization technology, cloud data centers host a wide range of Internet applications such as search engines, social media, video streaming, and very recently, train large generative AI models such as ChatGPT [24]. *Middleboxes* (MBs), on the other hand, have been an indispensable component in cloud data centers to improve the security and performance of many of the above Internet applications [46]. MBs, also called *network functions*, are intermediary network devices (e.g., firewalls, load balancers, and deep packet inspection) that inspect, filter, and transform network traffic other than packet forwarding [11].

To achieve the desired security and performance guarantees for the cloud applications, data center operators usually establish *data center policies* (or *service function chaining*) that require virtual machine (VM) traffic to traverse a chain of MBs of various security and performance functions [5], [38]. Fig. 1(a) shows such an example of a data center policy. Traffic generated at VM vm_1 goes through a firewall, a load balancer, and a cache proxy before arriving at VM vm'_1. In doing so, this policy filters out malicious VM traffic, diverts trusted VM traffic to avoid network congestion, and finally caches the traffic packets to share with other cloud users. We refer to cloud data centers that implement such security and performance policies as *policy-aware data centers* (PADCs).

Fig. 1(b) shows a small PADC that implements the same data center policy in Fig. 1(a). In this PADC, there are four physical machines (PMs) viz. pm_1, ..., pm_4, and five switches viz. s_1, ..., s_5. The firewall, load balancer, and cache proxy are installed on switches s_2, s_3, and s_4, respectively. Considering that PMs have limited *cloud resources* (i.e., CPU, memory, and disk I/O), and for illustration purposes, we assume that each PM can store one VM in this example.

Motivation. Meanwhile, measurements from Facebook and other production data centers show that *traffic rates* (e.g., communication frequencies or bandwidth demands) of communicating VM applications are highly diverse and dynamic [43]. One such example is cloud chatbots (e.g., Slack [4] and Amazon Lex [1]), wherein VMs can send low-bandwidth texts to each other at one instant, then switch to high-bandwidth live-streaming videos at another, and vice versa, while the duration of each communication varies considerably.

Therefore, it is important to design resource-efficient VM placement and migration for dynamic cloud data centers. This is especially crucial for PADCs with dynamic traffic – as the VM communication must go through a sequence of MBs in some specific order, the dynamic VM traffic rates could cause more network traffic and consume more *network resources* (i.e., bandwidths and energy consumption) in the PADC compared to traditional cloud data centers without such policies implemented.

Policy-Aware VM Placement and Migration. In this paper, we identify, formulate, and solve two new VM placement and migration problems for PADCs with diverse and dynamic traffic. Below we illustrate them with an example.

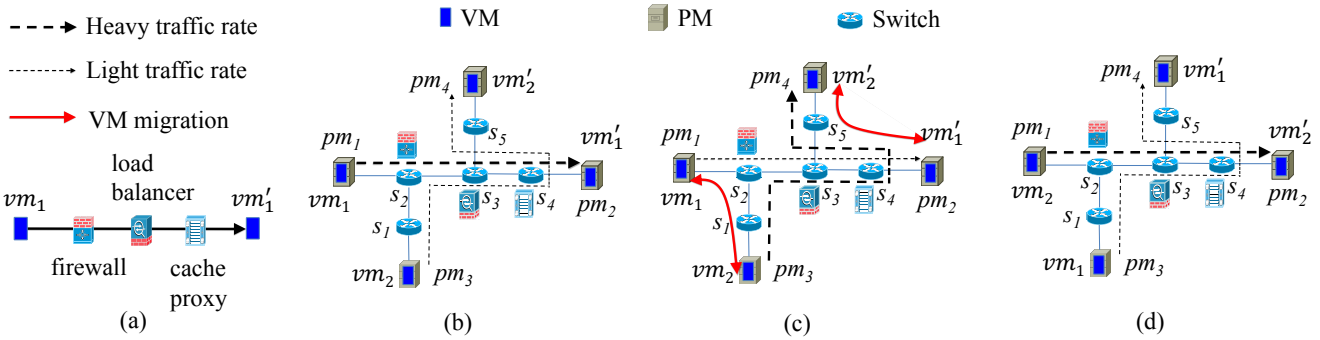


Fig. 1. (a) A data center policy in the PADC. (b) VM communication with the initial VM placement. (c) Dynamic VM traffic rates and the ensuing VM migration. (d) VM communication after the VM migration.

Policy-Aware VM Placement (PAL). Initially, when new cloud applications are launched as VMs in PADCs, the cloud operators need to decide where to place them for network-efficient communication. Given a PADC with a data center policy that communicating VM pairs must satisfy, PAL studies how to place the VMs with diverse traffic rates into the PMs to minimize their total communication cost while satisfying the resource constraints of PMs.

Fig. 1(b) shows two VM pairs (vm_1, vm'_1) and (vm_2, vm'_2), with the current traffic rate of the former much larger than that of the latter; we use dark-black and light-black dashed lines to indicate heavy traffic and light traffic, respectively. To reduce the traffic and delay of these two VM communication pairs while ensuring the VM traffic traverses the MBs in the order specified by the data center policy, it is preferred that vm_1 and vm'_1 (with a large traffic rate) communicate along a route that is "shorter" than that of vm_2 and vm'_2 (with a smaller traffic rate). Fig. 1(b) shows such an optimal VM placement solution of placing vm_1 on pm_1, vm'_1 on pm_2, vm_2 on pm_3, and vm'_2 on pm_4, with their *policy-aware communication routes* in dark-black and light-black dashed lines, respectively. With this placement, the communication route between vm_1 and vm'_1 has 4 network hops while the one between vm_2 and vm'_2 has 7 hops.

Policy-Aware VM Migration (PAM). Given an existing placement of communicating VM pairs with dynamic traffic rates, and a data center policy that they must satisfy, PAM studies how to migrate VMs to minimize the total cost of migration and communication of all the VM pairs.

Fig. 1(c) shows that the traffic rate of (vm_2, vm'_2) emerges as much heavier than that of (vm_1, vm'_1), due to dynamic traffic in PADCs. With new VM traffic rates, the initial VM placement in Fig. 1(b) is no longer communication inefficient – as vm_2 communicates with vm'_2 via a route much longer than that of (vm_1, vm'_1) (i.e., 7 hops verse 4 hops), it generates more network traffic in the PADC and consumes much of its network bandwidths.

Our key observation is that migrating VMs might be an effective technique to reduce network traffic and its corresponding network resource consumption. The red solid lines in Fig. 1(c) show the routes along which these two VM pairs are migrated; that is, vm_1 is swapped with vm_2, and vm'_1 is swapped with vm'_2 (recall each PM can only store one VM in the example). Fig. 1(d) shows the VM communication as the result of the VM migration, where the communication route of (vm_2, vm'_2) is shortened from 7 hops to 4 hops, greatly reducing the network traffic. Although the communication route of (vm_1, vm'_1) gets longer, as its traffic rate is small, the amount of increased traffic is much smaller than the amount of reduced traffic of (vm_2, vm'_2).

Our Contributions. Considering that the VM migration itself incurs network traffic and thus consumes network resources, and a large-scale PADC typically with tens of thousands of PMs and hundreds of thousands of communicating VMs with a wide range of changing traffic rates, how to place and migrate VMs in a large-scale PADC effectively becomes a challenging problem. In this paper, we address this challenge by formulating PAL and PAM and designing optimal, approximation, and heuristic *policy-aware* algorithms to solve them, while considering both uniform and heterogeneous resource demands of VMs. The PAM & PAL duo potentially achieves ideal resource utilization for a PADC's lifetime. After the PAL algorithms create the initial VM placement to optimize a PADC's initial cloud resource utilization, the PAM algorithms can then execute periodically to optimize a PADC's network resource utilization in response to dynamic VM traffic.

Using traffic patterns and flow characteristics observed in production data centers, we demonstrate that VM migration reduces the total communication costs of VM pairs by 25%, indicating that it is an effective technique for alleviating dynamic VM traffic in PADCs. We also show our *policy-aware* algorithms outperform the state-of-the-art policy-aware VM migration algorithm [16] by 30%, and the state-of-the-art VM placement algorithm that is oblivious to data center policies [41] by 40-50% under different PADC parameters. To the best of our knowledge, PAL and PAM are novel algorithmic frameworks that have not been adequately studied in existing literature.

Paper Organization. The paper is organized as follows. Section II surveys the related works, and Section III presents

the system model. Section IV and V formulate the PAL and PAM with uniform resource demands of the VMs and present their algorithms, respectively. Section VI extends it to a more general case wherein different VMs have different resource demands. Section VII discusses the simulation results and analysis. Section VIII concludes the paper with future works.

II. Related Work

A preliminary version of this paper appears as [28]. In this conference paper, we assume that all the VMs have uniform sizes, each consuming one resource slot. That is, all the VMs consume the same amount of resources (i.e., CPUs, memory, and disk I/O). Considering the diverse VM applications, not only could different communicating VM pairs have varying resource demands, but also the two VMs in the same pair could demand different amounts of resources. We thus extend our conference paper into the heterogeneous and more general case. We found that with heterogeneous resource demands of VMs, both PAL and PAM become more challenging. For example, under the ordered policy, while PAL and PAM with uniform VM sizes can both be solved optimally and efficiently, their heterogeneous counterparts become NP-hard and APX-hard, respectively. In this journal paper, we design approximation algorithms, integer linear programming models, and heuristic algorithms to address these problems.

Service function chaining. Service function chaining (SFC) has been an active research in recent years. It mainly focused on virtual network functions (VNFs) (i.e., virtual MBs) with their implementation and realization [33], [42], VNF placements [36], VNF migrations [48], [18], [39], and other issues such as availability [12], [26], flow control [45], and finding shortest SFC [44]. However, given that virtual MBs cannot yet match the performance of hardware MBs, many network functions still rely on dedicated hardware, and many hardware MBs still exist in enterprise networks [32], we consider hardware MBs in this paper. As such, once they are installed inside data centers, they cannot be easily moved around. However, our proposed approach still works with the PADC scenarios, wherein VNFs are fixed while VMs are placed and migrated.

VM Migration. There is a vast amount of literature on VM migration in cloud data centers [23], [55], [51], [53], [17], [21]. Laili et al. [23] aimed to reduce both overutilized and underutilized nodes, considering migration cost, communication overhead, and energy consumption, and proposed an iterative budget algorithm for dynamic VM consolidation in a cloud environment. Zhang et al. [55] analyzed the amount of bandwidth required to ensure the total migration time and downtime of a live VM migration. Wang et al. [51] studied how to migrate multiple VMs at the same time with available bandwidth, and designed a fully polynomial time approximation algorithm. Cui et al. [17] assumed that data center topologies are adaptive, featuring reconfigurable wireless links or optical circuit switches, and proposed a VM migration algorithm with a constant approximation ratio. Unlike other works, Liu [21] proposed a VM migration scheme to improve the reliability of cloud data centers. Please refer to [54] for a comprehensive review of VM migration literature.

VM migration studied in this paper, however, differs from the aforementioned works in both goals and models. While existing VM migration work has achieved various objectives, such as server consolidation, energy efficiency, load balancing, and fault tolerance in cloud data centers, our work focuses on reducing the network traffic caused by the dynamic traffic rates commonly found among communicating VMs. If not handled properly, dynamic traffic can significantly increase network traffic and waste network resources in cloud data centers. This problem is further exacerbated in PADCs by the fact that all the VM communication must follow the sequence of MBs required by the data center policies, generating more network traffic and consuming more resources. Besides, none of the above works considered data center policies, thus falling short of achieving the performance and security guarantees provided by various MBs deployed inside PADCs.

VM Placement. Meng et al. [41] designed one of the first *traffic-aware* VM placement algorithms by assigning VMs with large bandwidth usages to the host machines in close proximity. As PAL is one of the first to study *policy-aware* VM placement considering service function chaining thus no closely related work to compare with, we compare our work with this traffic-aware VM placement. Alicherry and Lakshman [8] designed optimal and approximation algorithms that place VMs to minimize data access latencies. Li et al. [37] studied VM placement to reduce the cost of the data center network and the cost of utilizing PMs. Wei et al. [52] considered unbalanced utilization of multi-resources (CPU, memory, storage, and network bandwidth), and presented a bi-objective optimization model for virtual machine placement. Again, they are policy-oblivious and thus do not achieve performance and security guarantees brought by PADCs. Note VM migration is different from the virtual network migration [56], [50]. The key problem in virtual network migration is to embed a sequence of virtual networks with both node and link constraints onto the physical network, which does not take consider the data center policy.

For the policy-aware VM migration, one closely related work to ours is by Cui et al. [16]. They proposed PLAN, the first policy-aware VM migration scheme for all-pair VM communications, and provided heuristic algorithms for ordered policies. In contrast, we focus on pairwise VM communication. They also assumed that the VM migration costs are constants that can be measured by the hypervisor hosting the VM. In contrast, our topology-aware migration cost attempts to capture the delay and bandwidth consumption of network traffic induced by VM migration. We design optimal, approximate, and heuristic algorithms for both ordered and unordered policies and show they all outperform PLAN.

One salient feature of our research is that it integrates VM migration and VM placement, the two most significant VM management mechanisms, into one framework. Existing VM

migration and placement research seems to achieve disparate goals – While VM placement optimizes *communication costs* among VMs such as energy cost, data access delay, and bandwidth usage [8], VM migration optimizes *migration costs* of VMs including migration time, downtime, and service degradation during migration [51], [17]. By modeling *topology-aware* VM migration and communication costs, we are able to jointly optimize VM placement and migration for the overall resource utilization in PADCs. We believe our work is the first to take a holistic approach to solving VM placement and migration in PADCs.

Reinforcement Learning Approach. In recent years, reinforcement learning has emerged as a powerful technique to solve dynamic networking problems in cloud data centers [19], [9], [25]. Chen et al. [19] proposed a fault tolerance-based SFC optimization in an SDN/NFV-enabled cloud environment based on deep reinforcement learning. Han et al. [25] aimed to optimize a weighted sum of the power consumption and resource shortage in data centers and proposed MadVM, an approximate Markov Decision Process. It achieved a VM migration cost of at most twice the optimal. Its scalability issue is addressed by a dimensionality reduction scheme [9] and a combined trend analysis employing the past, present, and future resource utilization [22]. Zeng et al. [20] proposed an adaptive deep reinforcement learning (DRL)-based VM consolidation framework for energy-efficient cloud data centers. Tuli [49] focused on node failure in containerized edge deployments and used a Generative Adversarial Network to predict preemptive migration decisions for proactive fault tolerance.

All the above works except [49] are centered on the dynamic resource demands of the VMs and the underload and overload of the PMs that host the VMs. Their goals are to migrate the VMs from underloaded PMs to overloaded PMs. In contrast, our work focuses on migrating communicating VMs with dynamic traffic rates to minimize the total network traffic (communication and migration) in cloud data centers.

III. System Models

Network Model. We model a PADC as an undirected general graph $G(V, E)$. $V = V_p \cup V_s$ includes a set of PMs $V_p = \{pm_1, pm_2, ..., pm_{|V_p|}\}$ and a set of switches $V_s = \{sw_1, sw_2, ..., sw_{|V_s|}\}$; E is a set of edges, each connecting either one switch to another switch or a switch to a PM. Fig. 2 shows a $k = 4$ fat-tree data center [7] where k is the number of ports each switch has. We use fat-tree to illustrate the problems and their algorithmic solutions. However, our problems and techniques are applicable to any data center topology. As live VM migration is typically performed within data center boundaries [13], we focus on VM placement and migration within a single data center in this paper.

There are a set of n MBs $\mathcal{M} = \{mb_1, mb_2, ..., mb_n\}$ inside the PADC, with mb_j installed at switch $sw_j \in V_s$. We adopt the *bump-off-the-wire* design [34], which uses a policy-aware switching layer to redirect traffic to off-path MBs explicitly. Fig. 2 shows three MBs mb_1, mb_2 and mb_3 installed using this design. As a switch and its attached MB are connected by high-speed optical fibers, the delay between them is negligible compared to that among switches and PMs [30].

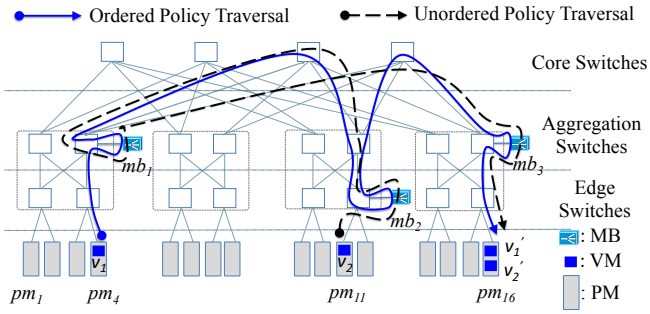


Fig. 2. A PADC with 16 PMs: pm_1, pm_2, ..., and pm_{16}, 3 MBs: mb_1, mb_2, and mb_3, and two VM pairs: (v_1, v_1') and (v_2, v_2'). • and ▶ indicate source and destination VM respectively.

There are l pairs of communicating VMs $\mathcal{P} = \{(v_1, v_1'), (v_2, v_2'), ..., (v_l, v_l')\}$ that are already placed into the PMs. For any VM pair (v_i, v_i'), $1 \le i \le l$, v_i and v_i' are referred to as its *source* and *destination VM* respectively. Denote the *traffic rate* of (v_i, v_i') as λ_i, and the *traffic rate vector* as $\overrightarrow{\lambda} = \langle \lambda_1, \lambda_2, ..., \lambda_l \rangle$; the traffic rate of each VM pair is their current communication frequencies or bandwidth demands. In a dynamic PADC, $\overrightarrow{\lambda}$ is not a constant vector as the VM traffic rates change over time. In Fig. 2, there are two VM pairs: (v_1, v_1') and (v_2, v_2'), with initial $\overrightarrow{\lambda} = \langle 100, 1 \rangle$.

Let $\mathcal{V} = \{v_1, v_1', v_2, v_2', ..., v_l, v_l'\}$. We denote the *resource capacity* of PM $pm_i \in V_p$ as rc_i, meaning that pm_i has rc_i *resource slots*. Here, the *resource* is an aggregated characterization of a PM's hardware resources such as CPUs, memories, and disk I/O. In Section IV and V, we assume that each VM $v \in \mathcal{V}$ consumes one source slot, and in Section VI, we consider the more general case of VMs with heterogeneous resource demands. Table I shows all the notations.

Cost Model. We model the VM communication and migration cost as either the delay or energy cost of the network traffic inside PADCs. We adopt a *topology-aware* model [41] and define the *communication cost* of any VM pair as the number of network links (i.e., hops) its traffic traverses multiplied by its traffic rate (however, our problems and solutions still hold for different edges that have different costs). The *migration cost* of migrating any VM v from PM i to PM j is $\mu \cdot c(i, j)$. Here, $c(i, j)$ is the minimum number of hops between any PM (or switch) i and j, and μ is a *migration coefficient* that depends on VM sizes and available network bandwidths.

Justifications. Our VM migration model is different from the one adopted by many existing literature. Mann et al. [40] focused on *pre-copy* [13], one of the original live VM migration techniques, and modeled the cost of migrating a VM v as $M_s \cdot \frac{1-(P_r/B_a)^{n+1}}{1-(P_r/B_a)}$. Here M_s is the image size of v, P_r is its page dirty rate, B_a is the available bandwidth,

TABLE I
NOTATION SUMMARY

Notation	Description
V_p	$V_p = \{pm_1, pm_2, ..., pm_{\lvert V_p \rvert}\}$ is the set of $\lvert V_p \rvert$ PMs
V_s	$V_s = \{sw_1, sw_2, ..., sw_{\lvert V_s \rvert}\}$ is the set of $\lvert V_s \rvert$ switches
$\mathcal{M}$	$\mathcal{M} = \{mb_1, mb_2, ..., mb_n\}$ is the set of n MBs
$\mathcal{P}$	$\mathcal{P} = \{(v_i, v_i'), ..., (v_l, v_l')\}$ is the set of l VM pairs
$\mathcal{V}$	$\mathcal{V} = \{v_1, ..., v_l, v_1', ..., v_l'\}$ are the source and dest. VMs
d_v	Resource demand of VM $v \in \mathcal{V}$ in heterogenous case
λ_i	Traffic rate between v_i and v_i', $1 \leq i \leq l$
$\overrightarrow{\lambda}$	$\overrightarrow{\lambda} = \langle \lambda_1, \lambda_2, ..., \lambda_l \rangle$
rc_i	Resource capacity of PM pm_i, $1 \leq i \leq \lvert V_p \rvert$
sw_j	Switch where mb_j is installed, $1 \leq j \leq n$
$c(i,j)$	Communication cost between PMs (or switches) i and j
$p(v)$	PM where the VM v is placed with VM placement p
π^i	Order at which (v_i, v_i') visits MBs in unordered policy
$\overrightarrow{\pi}$	$\overrightarrow{\pi} = \langle \pi^1, \pi^2, ..., \pi^l \rangle$
$C_c(p)$	Total communication cost with p in ordered policy
$C_c(p, \overrightarrow{\pi})$	Total communication cost with p in unordered policy
μ	VM migration coefficient
$m(v)$	PM where the VM v migrates to under VM migration m
$C_m(m)$	Total migration cost of VM migration m
$C_c(m)$	Total communication cost after VM migration m
$C_t(m)$	Total migra. and comm. cost with m in ordered policy
$C_t(m, \overrightarrow{\pi})$	Total migra. and comm. cost with m in unordered policy

and n is the number of pre-copy phases. They suggested the migration cost be a constant quantity measured by the hypervisor. In contrast, by acknowledging network delay or energy consumption incurred by VM migration traffic, our topology-aware model is more conducive to designing VM migration algorithms for a large-scale and dynamic traffic environment targeted by this paper.

Ordered and Unordered Data Center Policies. Depending on the application requirements, some policies require that the VM traffic go through the MBs in a strict order. We refer to such policies as *ordered policies* and denote them as $(mb_1, mb_2, ..., mb_n)$. On the other hand, as MB functions are mostly independent, some data center policies are satisfied as long as the VM traffic visits all its MBs. We refer to such policies as *unordered policies* and denote them as $\{mb_1, mb_2, ..., mb_n\}$. In Fig. 2, (v_1, v_1') traverses MBs under ordered policy (mb_1, mb_2, mb_3), resulting in communication cost of $100 \times 10 = 1000$ (solid blue line), (v_2, v_2') traverses MBs under unordered policy $\{mb_1, mb_2, mb_3\}$, resulting in communication cost of $1 \times 8 = 8$ (dashed black line).

We refer to the first (and last) visited MB in a policy as *ingress (and egress) MB*, and the switch where the ingress (and egress) MB is installed as *ingress (and egress) switch*. For the ordered policy, the ingress switch is sw_1 and the egress switch is sw_n. For an unordered policy, it needs to determine sw_1 and sw_n as well as the MB sequence along which the VM pair communicates. As one data center policy shown in Fig. 1(a) is generally sufficient to serve both security and performance purposes, we assume there is one data center policy (ordered or unordered) in a PDDC at a time. We adopt FlowTags [27], an SDN architecture that provides consistent policy enforcement during VM migration by adding tags in packet headers.

EXAMPLE 1: Fig. 3 shows a $k = 2$ fat tree PADC with two PMs: pm_1 and pm_2, where pm_1 has two resource slots rs_1 and rs_2 and pm_2 has rs_3 and rs_4. Two MBs, mb_1 and mb_2, are installed on edge switch sw_1 and aggregation switch sw_2, respectively. There are two VM pairs (v_1, v_1') and (v_2, v_2'), v_1 and v_2 are placed on pm_1 while v_1' and v_2' on pm_2. $\overrightarrow{\lambda} = \langle 100, 1 \rangle$ and $\mu = 1$. Before VM migration, the total communication cost of the two VM pairs in Fig. 3(a) is 606 under both the ordered policy (mb_1, mb_2) and the unordered policy $\{mb_1, mb_2\}$. By migrating v_1' to pm_1 and v_2 to pm_2 (with a migration cost of 12), shown in the solid red line in Fig. 3(a), the total cost of the VM communication (dotted and dashed black lines in Fig. 3(b)) becomes 410. This is a 30% reduction in total cost compared to before migration. We prove this VM migration is optimal in Section V-A. □

IV. PAL: POLICY-AWARE VM PLACEMENT IN PADCS

Section IV and V consider the uniform case wherein each VM consumes one unit of resource slot, and study PAL and PAM, respectively. As there are $2 \cdot l$ VMs and each needs one resource slot, it must be that $\sum_{i=1}^{\lvert V_p \rvert} rc_i \geq 2 \cdot l$. Given a PADC with a set of communicating VM pairs of varying traffic rates and a data center policy to satisfy, PAL studies how to place the VMs within the PADC to minimize their total communication cost while satisfying the resource constraints of PMs. Below, we formulate PAL under ordered and unordered policies and design optimal and approximation algorithms.

A. Ordered Policy.

1) Problem Formulation: Under the ordered policy, for any VM pair communication, the ingress switch is always sw_1 and the egress switch is always sw_n. Given a VM placement function p, denote the *total communication cost* of all the l VM pairs under p as $C_c(p)$. We have

$$C_c(p) = \sum_{i=1}^{l} \lambda_i \cdot \sum_{j=1}^{n-1} c\big(sw_j, sw_{j+1}\big) + \sum_{i=1}^{l} \lambda_i \cdot \Big(c\big(p(v_i), sw_1\big) + c\big(sw_n, p(v_i')\big) \Big). \tag{1}$$

Given any VM placement p, for any VM pair (v_i, v_i'), we refer to $p(v_i)$ and $p(v_i')$ as its *source* and *destination PM* respectively. The objective of PAL is to find a VM placement p to minimize $C_c(p)$ while satisfying the resource constraint of PMs: $|\{v \in \mathcal{V} | p(v) = i\}| \leq rc_i, \forall i \in V_p$. As the first term on the r.h.s. of Eq. 1 is fixed under the ordered policy, we only need to minimize the second term. Below, we design an optimal and efficient algorithm to solve the PAL problem.

2) VM Placement Algorithm for Ordered Policy: To save communication costs for VM pairs, the key is to find PMs close to the ingress (and egress) switch, and then to place source (and destination) VMs into their respective resource slots. We give the definitions below.

Definition 1: (**Ingress/Egress Costs, Ingress/Egress Resource Sets, Optimal Ingress/Egress Sets**) A resource slot rs's *ingress* (and *egress*) cost, denoted as $c_{in}(rs)$ (and $c_e(rs)$), is the cost between its source PM (and destination PM) and the ingress switch sw_1 (and egress switch sw_n). Let

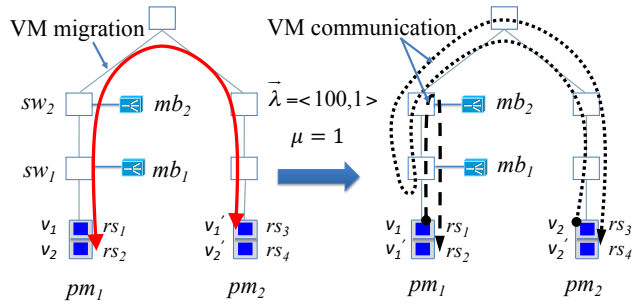


Fig. 3. VM migration achieved 30% of total cost reduction in a linear PADC.

$pm(rs)$ be the PM rs belongs to, $c_{in}(rs) = c\big(pm(rs), sw_1\big)$, $c_e(rs) = c\big(pm(rs), sw_n\big)$.

An *ingress* (and *egress*) *resource set* (IRS and ERS) is a set of l resource slots that store the l source (and destination) VMs. The *cost* of an IRS (and ERS) is the sum of the ingress (and egress) costs of its resource slots. A pair of IRS and ERS is *optimal*, denoted as (I^{opt}, E^{opt}), if the sum of their costs is the minimum among all pairs of IRS and ERS. □

I^{opt} and E^{opt} are data structures that uniquely arise in PAL. Below, we present Algo. 1 to find I^{opt} and E^{opt} and Theorem 1 to prove that they are optimal IRS and ERS achieving minimizing communication cost for the VM pairs.

Algo. 1 begins by sorting the resource slots in the non-descending order of their ingress (and egress) costs and storing the top $2l$ resource slots in an array $\mathcal{I}$ (and an array $\mathcal{E}$) (line 2). It then scans through $\mathcal{I}$ and $\mathcal{E}$ and finds the next available resource slot in each, and checks if they are distinct (lines 4-6). If so, these two slots are selected as the next pair of ingress and egress resource slots in I^{opt} and E^{opt} (lines 7-9). Otherwise, with one resource slot found, it identifies the other one from either $\mathcal{I}$ or $\mathcal{E}$ with a lower cost (lines 11-20). This takes place until l distinct pairs of ingress and egress resource slots are found, which are the I^{opt} and E^{opt} (lines 21-22). For all the l VM pairs (in non-ascending order of their traffic rates), it then places their source VMs into I^{opt} and destination VMs into E^{opt} while recording the placement p and calculating the total communication cost $C_c(p)$ along the way (lines 23-29). Finally, it returns the obtained p and $C_c(p)$ (line 30).

Algorithm 1: PAL Algorithm for Ordered Policy.
Input: A PADC with ordered policy $(mb_1, mb_2, ..., mb_n)$, VM pairs $\mathcal{P}$, $V_p = \{pm_i\}$, resource capacity rc_i of pm_i.
Output: A VM placement p and the total communication cost of all VM pair $C_c(p)$.
Notations: $\mathcal{I}, \mathcal{E}$: the ingress and egress slots, each of size $2l$.
I^{opt}, E^{opt}: the optimal ingress and egress slots, each of size l.
i, j: indices for slots in $\mathcal{I}$ and $\mathcal{E}$ respectively.
k: index for slots in I^{opt} and E^{opt}.
$sel(rs_i)$: initially *false*, *true* if rs_i is put into I^{opt} or E^{opt}.
1. $i = j = k = 1$, $C_c(p) = 0$, $p = I^{opt} = E^{opt} = \phi$ (empty set);
2. Sort resource slots in non-descending order of their ingress (and egress) costs, store the top $2l$ in arrays $\mathcal{I}$ (and $\mathcal{E}$);
3. **while** $(k \le l)$ // find optimal resource slots for (v_k, v_k')
4. **if** $(sel[\mathcal{I}[i]] == true)$ i++;
5. **if** $(sel[\mathcal{E}[j]] == true)$ j++;
6. **if** $(\mathcal{I}[i] \neq \mathcal{E}[j]$) // both optimal resource slots are found
7. $I^{opt}[k] = \mathcal{I}[i]$, $E^{opt}[k] = \mathcal{E}[j]$;
8. $sel[\mathcal{I}[i]] = sel[\mathcal{E}[j]] = true$;
9. i++, j++;
10. **else** // one found, now find the other
11. **if** $(c_{in}(\mathcal{I}[i]) + c_e(\mathcal{E}[j+1]) \le c_{in}(\mathcal{I}[i+1]) + c_e(\mathcal{E}[j]))$
12. $I^{opt}[k] = \mathcal{I}[i]$, $E^{opt}[k] = \mathcal{E}[j+1]$;
13. $sel[\mathcal{I}[i]] = sel[\mathcal{E}[j+1]] = true$;
14. i++, j += 2;
15. **else**
16. $I^{opt}[k] = \mathcal{I}[i+1]$, $E^{opt}[k] = \mathcal{E}[j]$;
17. $sel[\mathcal{I}[i+1]] = sel[\mathcal{E}[j]] = true$;
18. i += 2, j++;
19. **end if;**
20. **end if;**
21. k++;
22. **end while;**
23. Sort VM pairs in non-ascending order of their traffic rates, assume $\lambda_1 \ge \lambda_2 ... \ge \lambda_l$;
24. **for** $(1 \le i \le l)$ // place VM pairs and calculate cost
25. Place v_i at $I^{opt}[i]$ and v_i' at $E^{opt}[i]$;
26. $p = p \cup \{(I^{opt}[i], E^{opt}[i])\}$;
27. $C_c(p)$ += $\lambda_i * \big(c_{in}(I^{opt}[i]) + c_e(E^{opt}[i])\big)$;
28. **end for;**
29. $C_c(p)$ += $\sum_{i=1}^{l} \lambda_i \sum_{j=1}^{n-1} c\big(sw_j, sw_{j+1}\big)$;
30. **RETURN** p and $C_c(p)$.

Finding $\mathcal{I}$ and $\mathcal{E}$ takes $O(|V_p| \cdot \bar{m} \cdot \lg(|V_p| \cdot \bar{m}))$, where $|V_p|$ is the number of PMs and $\bar{m} = \sum_{i=1}^{|V_p|} rc_i/|V_p|$ is the average resource capacity of PMs. Finding I^{opt} and E^{opt} takes l rounds, each could take $O(l)$. Sorting all the VM pairs takes $O(l \cdot \lg l)$ and calculating $C_c(p)$ takes $O(l)$. As l is bounded by $(|V_p| \cdot \bar{m})$, the time complexity of Algo. 1 is $O(|V_p|^2 \cdot \bar{m}^2)$.

EXAMPLE 2: Fig. 4(a) shows how Algo. 1 works to place the two VM pairs (v_1, v_1') and (v_2, v_2') into the PADC in Fig. 3. It first finds $\mathcal{I}$ and $\mathcal{E}$, which are both $\{rs_1, rs_2, rs_3, rs_4\}$. It then compute $I^{opt} = \{rs_1, rs_3\}$ and $E^{opt} = \{rs_2, rs_4\}$. Following this computation, it places v_1 into rs_1 and v_1' into rs_2, and v_2 into rs_3 and v_2' into rs_4. The total communication cost of this placement is $100 \cdot 4 + 1 \cdot 10 = 410$. Other placements result in a larger total communication cost. For example, placing v_1 and v_2 in pm_1 and v_1' and v_2' in pm_2 (shown in Fig. 3(a)) results in cost of $100 \cdot 6 + 1 \cdot 6 = 606$. As rs_1 and rs_2 are both in pm_1 and rs_3 and rs_4 are both in pm_2, this placement is exactly the same as the one in Fig. 3(b). □

Lemma 1: (I^{opt}, E^{opt}) computed in Algo. 1 (lines 1-22) is a pair of optimal IRS and ERS.

Proof: First, we show that $\mathcal{I}$ (and $\mathcal{E}$) obtained from Algo. 1 contains at least one IRS (and one ERS). This is because among the $2l$ resource slots in $\mathcal{I}$, at most l of them could belong to an ERS, leaving at least l resource slots that can be allocated for an IRS. Thus, $\mathcal{I}$ contains at least one IRS. A

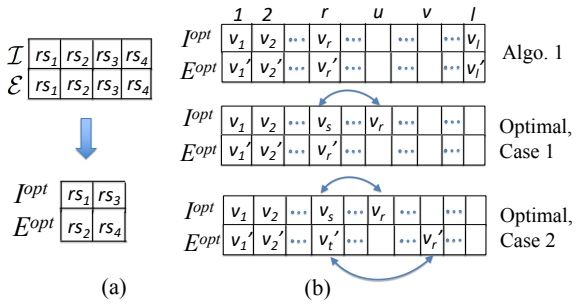


Fig. 4. (a) A working example and (b) optimality proof for Algo. 1.

similar argument works for $\mathcal{E}$.

As $I^{opt} \subset \mathcal{I}$ and $E^{opt} \subset \mathcal{E}$, next, we show that (I^{opt}, E^{opt}) is a pair of optimal IRS and ERS. We prove by induction. When $k = 1$, if $\mathcal{I}[1] \neq \mathcal{E}[1]$, $c_{in}(I^{opt}[1]) + c_e(E^{opt}[1]) = c_{in}(\mathcal{I}[1]) + c_e(\mathcal{E}[1])$; if $\mathcal{I}[1] == \mathcal{E}[1]$, $c_{in}(I^{opt}[1]) + c_e(E^{opt}[1]) = \min\big(c_{in}(\mathcal{I}[1]) + c_e(\mathcal{E}[2]), c_{in}(\mathcal{I}[2] + c_e(\mathcal{E}[1]))\big)$. Either way, $c_{in}(I^{opt}[1]) + c_e(E^{opt}[1])$ is minimum, as $\mathcal{I}$ and $\mathcal{E}$ store resource slots in a non-descending order of ingress and egress costs, respectively. Now assume for any $1 \leq q < l$, $\sum_{k=1}^{q} \big(c_{in}(I^{opt}[i]) + c_e(E^{opt}[i])\big)$ is minimum. The unscanned part of $\mathcal{I}$ and $\mathcal{E}$ are $\mathcal{I}[i], \mathcal{I}[i+1]$, ..., $\mathcal{I}[l]$ and $\mathcal{E}[j], \mathcal{E}[j+1]$, ..., $\mathcal{E}[l]$, respectively, where $i, j \geq q$. If $\mathcal{I}[i] \neq \mathcal{E}[j]$, $c_{in}(I^{opt}[q+1]) + c_e(E^{opt}[q+1]) = c_{in}(\mathcal{I}[i]) + c_e(\mathcal{E}[j])$; if $\mathcal{I}[i] == \mathcal{E}[j]$, $c_{in}(I^{opt}[q+1]) + c_e(E^{opt}[q+1]) = \min\big(c_{in}(\mathcal{I}[i]) + c_e(\mathcal{E}[j+1]), c_{in}(\mathcal{I}[i+1] + c_e(\mathcal{E}[j]))\big)$. Either way $c_{in}(I^{opt}[1+1]) + c_e(E^{opt}[q+1])$ is minimum among the rest of unscanned pairs. Thus $\sum_{k=1}^{q+1} \big(c_{in}(I^{opt}[i]) + c_e(E^{opt}[i])\big)$ is minimum, proving that (I^{opt}, E^{opt}) is a pair of optimal IRS and ERS. ∎

Theorem 1: Algo. 1 is optimal. That is, given any PAL instance, it always finds the VM placement that minimizes total communication cost for the l VM pairs.

Proof: As Lemma 1 proves that (I^{opt}, E^{opt}) is a pair of optimal IRS and ERS, it is left to show that the VM placement on I^{opt} and E^{opt} in Algo. 1 (lines 23-30) yields minimum total communication cost for the l VM pairs. We prove by contradiction and assume that Algo. 1 is not optimal. Instead, another algorithm, named Optimal, provides an optimal solution. Therefore, there must exist an r, $1 \leq r \leq l$, that is the smallest index at which $I^{opt}[r]$ or $E^{opt}[r]$ stores a different pair of VMs for Algo. 1 and Optimal. Two cases are shown in Fig. 4(b). Case 1: one of the two resource slots, $I^{opt}[r]$ or $E^{opt}[r]$, stores different VMs. For example, both algorithms stores v_r' at $E^{opt}[i]$ while Algo. 1 stores v_r and Optimal stores v_s at $I^{opt}[i]$. Case 2: both slots store different VMs. For example, Optimal stores v_s at $I^{opt}[r]$ and v_t' at $E^{opt}[r]$ (with $s, t > r$), and stores v_r at $I^{opt}[u]$ and v_r' at $E^{opt}[v]$ (with $u, v > r$). In both cases, we can swap VMs in Optimal following blue curved arrow lines to further reduce its cost, due to $\lambda_1 \geq \lambda_2 ... \geq \lambda_l$. ∎

B. Unordered Policy.

1) Problem Formulation: In unordered policy, besides a VM placement function p, PAL needs to find an order for *each* VM pair to visit the MBs (note that different VM pairs could traverse the MBs in different orders). For (v_i, v_i') we define such order as an MB *traversal function* $\pi^i : [1, 2, ..., n] \rightarrow [1, 2, ..., n]$, a permutation function indicating the j^{th} MB that (v_i, v_i') visits is $mb_{\pi^i(j)}$. Given p and π^i, denote (v_i, v_i')'s communication cost as c_i^{p,π^i}. Then c_i^{p,π^i}

$$= \lambda_i \cdot c\Big(p(v_i), sw\big(\pi^i(1)\big)\Big) + \lambda_i \cdot c\Big(sw\big(\pi^i(n)\big), p(v_i')\Big) + \lambda_i \cdot \sum_{j=1}^{n-1} c\Big(sw\big(\pi^i(j)\big), sw\big(\pi^i(j+1)\big)\Big). \quad (2)$$

Here the first two terms are the communication cost from the source VM v_i to the ingress switch $sw\big(\pi^i(1)\big)$ and destination VM v_i' to the egress switch $sw\big(\pi^i(n)\big)$, respectively, and the third term is their cost between ingress and egress switches. Let $\overrightarrow{\pi} = \langle \pi^1, \pi^2, ..., \pi^l \rangle$. The objective of PAL under unordered policy is to minimize *total communication cost* $C_c(p, \overrightarrow{\pi}) = \sum_{i=1}^{l} c_i^{p,\pi^i}$ while satisfying $|\{v \in \mathcal{V} | p(v) = i\}| \leq rc_i, \forall i \in V_p$. Below, we show that PAL is NP-hard even for one pair of VMs. We then propose an approximation algorithm for this special case that yields a total cost at most twice that of the optimal.

Theorem 2: Under unordered policy, PAL is NP-hard even for one pair of VMs (v_1, v_1') (i.e., $l = 1$).

Proof: We reduce *s-t traveling salesman path problem* (TSPP) [31], which is NP-hard, to this problem. Given a complete undirected graph $K = (V_K, E_K)$ with edge cost $d : E_K \rightarrow \mathbb{R}^+$ and a pair of pre-specified vertices $s, t \in V_K$, TSPP finds a shortest *Hamiltonian path* that starts at s, visits each vertex in $V_K - \{s, t\}$ exactly once, and ends at t. d satisfies *triangle inequality*, i.e., $d(u, w) \leq d(u, v) + d(v, w)$ for all $u, v, w \in V_K$, When $s = t$, TSPP becomes well-known *traveling salesman problem* (TSP) [15], which finds a shortest *Hamiltonian cycle*.

Given VM pair (v_1, v_1') and an instance of PADC graph $G(V = V_p \cup V_s, E)$, we construct $|V_p| \cdot (|V_p| + 1)/2$ instances of complete graphs $K^{i,j} = (V_K^{i,j}, E_K^{i,j})$, $1 \leq i \leq |V_p|$, $i \leq j \leq |V_p|$. Here, $V_K^{i,j} = \{pm_i, pm_j, sw_1, sw_2, ...sw_n\}$ and for $(u, v) \in E_K^{i,j}$, its cost $d(u, v)$ is the cost of the shortest path connecting u and v in G. Now, if $K^{a,b}$ has the shortest Hamiltonian path whose cost is the minimum among the shortest Hamiltonian paths in all the instances, then placing v_1 to pm_a and v_1' to pm_b must give the minimum communication cost for (v_1, v_1') in G, and vice versa. ∎

2) VM Placement Algorithm for Unordered Policy: We first give the definition below before presenting the algorithms.

Definition 2: (**Optimal Policy Route (OPR).**) In a PADC graph, a *policy route* of any pair of PMs (pm_i, pm_j) is a path or walk starting pm_i, visiting all the n MBs at least once, and ending at pm_j. An OPR of (pm_i, pm_j) gives the minimum cost, denoted as $opr(i, j)$, among all its policy routes. □

OPR of (pm_i, pm_j) is essentially the shortest *s-t* Hamiltonian path in the aforementioned TSPP [31], with $s = pm_i$ and $t = pm_j$ in a complete graph of pm_i, pm_j and all MBs (when $pm_i = pm_j$, it is a Hamiltonian cycle). The state-of-the-art work solving TSPP [31] follows the well-known Christofides' algorithm [2] of the traveling salesman problem and achieves

an approximation ratio of $\frac{5}{3}$ and takes $O(n^3)$, where n is the number of MBs. Christofides' algorithm first finds a minimum spanning tree (MST) of G, and then traverses on a minimum-weight perfect matching in the induced subgraph given by the set of vertices with odd degrees in the MST. Below, we instead propose another $O(n^3)$ algorithm to compute a policy route for (pm_i, pm_j). Our algorithm is much easier to implement and has an approximation ratio of 2.

Algorithm 2: Compute a Policy Route for (pm_i, pm_j).
Input: A PADC graph G, a pair of PMs (pm_i, pm_j), and an unordered policy $\{mb_1, mb_2, ..., mb_n\}$.
Output: $pr(i,j)$, cost of a policy route for (pm_i, pm_j).
1. $V_K^{i,j} = \{pm_i, pm_j, sw_1, sw_2, ...sw_n\}$;
2. Construct complete graph $K^{i,j} = (V_K^{i,j}, E_K^{i,j})$. For edge $(u,v) \in E_K^{i,j}$, its cost $d(u,v)$ is the cost of the shortest path connecting u and v in G;
3. Compute a minimum spanning tree MST for $K^{i,j}$;
4. Compute a walk W from pm_i to pm_j on MST that visits all vertices in MST using each edge *at most twice*. Let the cost of W be $pr(i,j)$;
5. **RETURN** $pr(i,j)$.

Using Algo. 2, Fig. 5 shows in blue dashed lines all three possible policy routes in the linear PADC of Fig. 3.

Lemma 2: $pr(i,j) \leq 2 \cdot opr(i,j)$, $\forall pm_i, pm_j \in V_p$.
Proof: Denote the MST cost computed in line 3 of Algo. 2 as c(MST), c(MST) $\leq opr(i,j)$. Since the walk W found in line 4 uses each edge of the MST at most twice, $pr(i,j) \leq 2 \cdot c$(MST). Thus, we have $pr(i,j) \leq 2 \cdot opr(i,j)$. ∎

Next, we present our PAL algorithm Algo. 3 for unordered policy. It first computes the policy routes for all the $|V_p| \cdot (|V_p|+1)/2$ pairs of PMs using Algo. 2 and orders them in the non-descending order of their costs (lines 1-8). It then places the VM pairs (in the non-ascending order of their traffic rates) onto the first available PM pair, and updates the total communication cost accordingly (lines 9-21). Running time of Algo. 3 is $O(|V_p|^2 \cdot n^3 + l)$.

Algorithm 3: PAL Algorithm for Unordered Policy.
Input: A PADC with unordered policy $\{mb_1, mb_2, ..., mb_m\}$, VM pairs $\mathcal{P}$, $V_p = \{pm_i\}$, resource capacity rc_i of pm_i.
Output: A placement p and the total comm. cost $C_c(p, \overrightarrow{\pi})$.
Notations: X: all PM pairs with their policy route costs.
$avail(pm_i)$: available resource slots at pm_i, initially rc_i.
1. $X = \phi$; $avail(pm_i) = rc_i$, $\forall pm_i \in V_p$;
2. **for** $(i = 1; i \leq |V_p|; i{+}{+})$
3. **for** $(j = i; j \leq |V_p|; j{+}{+})$
4. Compute $pr(i,j)$ using Algo. 2;
5. $X = X \cup \{(i, j, pr(i,j))\}$;
6. **end for;**
7. **end for;**
8. Sort X in non-descending order of $pr(i,j)$. Let X be $\{(s_1, t_1, pr(s_1,t_1)), (s_2, t_2, pr(s_2,t_2)), ...\}$;
9. $i = 1$, $j = 1$, $p = \phi$, $C_c(p, \overrightarrow{\pi}) = 0$, $\lambda_1 \geq \lambda_2 ... \geq \lambda_l$;
// i, j: indices for VM pairs and PM pairs respectively.

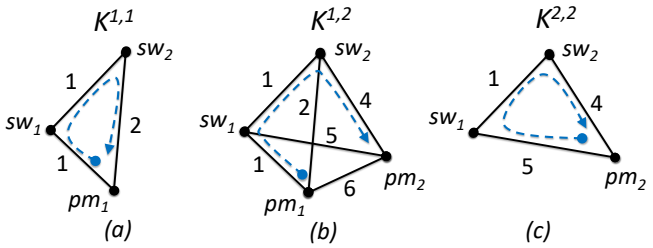


Fig. 5. How Algo. 3 works for PADC in Fig. 3. Blue dashed lines show the policy route of (a) (pm_1, pm_1), (b) (pm_1, pm_2), and (c) (pm_2, pm_2).

10. **while** $(i \leq l)$ // not all VM pairs are placed yet
11. **do**
12. Place v_i at PM s_j, v_i' at PM t_j;
13. $p = p \cup \{(s_j, t_j)\}$;
14. $C_c(p, \overrightarrow{\pi})$ += $\lambda_i * pr(s_j, t_j)$;
15. $avail(s_j)--$, $avail(t_j)--$;
16. $i{+}{+}$;
17. **if** $(i > l)$ **break**;
18. **while** $\big((s_j \neq t_j \wedge avail(s_j) \geq 1 \wedge avail(t_j) \geq 1) \vee (s_j == t_j \wedge avail(s_j) \geq 2)\big)$;
19. $j{+}{+}$; // the next available PM pairs
20. **end while;**
21. **RETURN** p and $C_c(p, \overrightarrow{\pi})$.

EXAMPLE 3: Fig. 5 shows how Algo. 3 places (v_1, v_1') and (v_2, v_2'), with $\overrightarrow{\lambda} = \langle 100, 1 \rangle$, in the linear PADC in Fig. 3. It computes $X = \{(1,1,4), (1,2,6), (2,2,10)\}$. Thus (v_1, v_1') is placed at pm_1 and communicates in blue dashed line in Fig. 5(a). As pm_1 is now full, (v_2, v_2') is placed at pm_2 and communicates in blue dashed line in Fig. 5(c). The total communication cost is $100 \cdot 4 + 1 \cdot 10 = 410$. □

Theorem 3: Algo. 3 achieves 2-approximation when $l = 1$.

Proof: Let the placement of (v_1, v_1') computed by Algo. 3 be (pm_a, pm_b). Let the optimal placement of (v_1, v_1') be $(pm_{a'}, pm_{b'})$ thus their optimal communication cost is $opr(a', b')$. From Lemma 2, we have $pr(a', b') \leq 2 \cdot opr(a', b')$. As the costs of all PM pair routes are sorted in non-descending order in Algo. 3, $pr(a,b) \leq pr(a', b') \leq 2 \cdot opr(a', b')$. ∎

V. PAM: Policy-Aware VM Migration in PADCs

Due to dynamic traffic in PADC, $\overrightarrow{\lambda}$ could constantly change. With such changes, the VM placement computed from Section IV may no longer be optimal or approximate solutions. To reduce communication costs, the VMs may need to migrate from their current PMs to other PMs, allowing them to communicate at a lower cost. As VM migration incurs costs in PADC, the objective of PAM is to migrate VMs in a way that minimizes the total cost of VM migration and VM communication. Below, we formulate and solve PAM in ordered and unordered policies, respectively.

A. Ordered Policy.

1) Problem Formulation: In PAM, the initial VM placement is given by a *placement function* $p : \mathcal{V} \to V_p$, indicating that VM $v \in \mathcal{V}$ is at PM $p(v) \in V_p$. The total communication cost of all the l VM pairs with placement p is thus $C_c(p)$ (Eq. 1). Next, define a VM *migration function* as $m : \mathcal{V} \to V_p$, meaning that VM v will be migrated from PM $p(v)$ to PM $m(v)$. Note that $m(v) = p(v)$ if v does not migrate. Let $C_m(m)$ be the *total migration cost* of all VM pairs with migration m. Then

$$C_m(m) = \mu \cdot \sum_{i=1}^{l} \Big(c\big(p(v_i), m(v_i)\big) + c\big(p(v_i'), m(v_i')\big) \Big). \quad (3)$$

Let $C_c(m)$ be the *total communication cost* of all VM pairs *after* migration m. Then

$$\begin{aligned} C_c(m) = &\sum_{i=1}^{l} \lambda_i \cdot \sum_{j=1}^{n-1} c\big(sw_j, sw_{j+1}\big) + \\ &\sum_{i=1}^{l} \lambda_i \cdot \Big(c\big(m(v_i), sw_1\big) + c\big(sw_n, m(v_i')\big) \Big). \end{aligned} \quad (4)$$

Let $C_t(m)$ be the *total migration and communication cost* of all pairs *after* m. Then,

$$\begin{aligned} C_t(m) &= C_m(m) + C_c(m) \\ &= \sum_{i=1}^{l} \lambda_i \cdot \sum_{j=1}^{n-1} c\big(sw_j, sw_{j+1}\big) \\ &+ \sum_{i=1}^{l} \Big(\mu \cdot c\big(p(v_i), m(v_i)\big) + \lambda_i \cdot c\big(m(v_i), sw_1\big) \Big) \\ &+ \sum_{i=1}^{l} \Big(\mu \cdot c\big(p(v_i'), m(v_i')\big) + \lambda_i \cdot c\big(sw_n, m(v_i')\big) \Big). \end{aligned} \quad (5)$$

The objective of PAM is to find a VM migration m that minimizes $C_t(m)$ while satisfying resource constraint of PMs: $|\{v \in \mathcal{V} | m(v) = pm_i\}| \leq rc_i, \forall pm_i \in V_p$. As the first term on the right-hand side in Eq. 5 is fixed under the ordered policy, we only need to minimize the sum of the last two terms. Below, we show that PAM under an ordered policy is equivalent to the minimum cost flow problem [6] in a flow network that is properly transformed from the PADC graph.

2) Minimum Cost Flow (MCF) Problem: MCF is formally defined as follows. Let $G = (V, E)$ be a directed graph. Denote the capacity and cost of an edge $(u, v) \in E$ as $ca(u, v)$ and $d(u, v)$, respectively. The amount of supply from source node $s \in V$ equals the amount of demand at sink node $t \in V$. Denote a flow on edge (u, v) as $f(u, v)$, $f : E \to \mathbb{R}^+$. $f(u, v)$ is subject to (a) capacity constraint: $f(u, v) \leq ca(u, v), \forall (u, v) \in E$ and (b) flow conservation constraint: $\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u)$, for each $v \in V \setminus \{s, t\}$. The goal of MCF is to find a flow function f such that the total cost of the flow $\Sigma_{(u,v) \in E}\big(d(u, v) \cdot f(u, v)\big)$ is minimized. MCF can be solved efficiently and optimally by many combinatorial algorithms [6]. In this paper, we adopt the scaling push-relabel algorithm proposed by Goldberg [29] as it works well over a wide range of problem classes. The algorithm has the time complexity of $O(A^2 \cdot B \cdot \log(A \cdot C))$, where A, B, and C are the number of nodes, number of edges, and maximum edge capacity in the flow network, respectively.

Transforming a PADC Graph to a Flow Network. We transform a PADC graph $G(V, E)$ into a flow network $G'(V', E')$ following below five steps, as shown in Fig. 6(a).

Step I. $V' = \{s\} \cup \{t\} \cup \mathcal{V} \cup V_p$, where s is the source node and t is the sink node in the flow network.

Step II. $E' = \{(s, v)\} \cup \{(v, pm_j)\} \cup \{(pm_j, t)\}$, where $v \in \mathcal{V}$ and $pm_j \in V_p$. Note that it is a complete bipartite graph between $\mathcal{V}$ and V_p.

Step III. For each edge (s, v), set its capacity as 1 and cost as 0. For each edge (pm_j, t), set its capacity as rc_j, the resource capacity of pm_j, and cost as 0.

Step IV. For each edge (v_i, pm_j), set its capacity as 1 and cost as $\mu \cdot c\big(p(v_i), pm_j\big) + \lambda_i \cdot c\big(pm_j, sw_1\big)$. For each edge (v_i', pm_j), set its capacity as 1 and cost as $\mu \cdot c\big(p(v_i'), pm_j\big) + \lambda_i \cdot c\big(pm_j, sw_n\big)$.

Step V. Set the supply at s and the demand at t as $2l$.

Theorem 4: PAM in ordered policy is equivalent to MCF in $G'(V', E')$ and thus can be solved optimally and efficiently.

Proof: We prove that by applying the MCF algorithm to the above flow network Fig. 6(a), it is able to achieve that a) every VM in the l VM pairs is assigned (i.e., migrated) to exactly one resource slot in a PM while b) satisfying the resource capacity constraints of PMs and c) achieving the minimum total migration and communication cost for all the l VM pairs.

First, as the supply at s is $2l$ and the capacity of each of the $2l$ edges (s, v), $v \in \mathcal{V}$, is 1, there must be one unit of flow going from s to each v. Due to flow conservation, there must be one unit of flow leaving v to reach one PM, pm_j. This indicates that each VM v will be migrated to one PM.

Second, as the capacity rc_j of an edge (pm_j, t) is the resource capacity of pm_j, there cannot be more than rc_j amount of flow from pm_j to t. Consequently, there cannot be more than rc_j amount of flow coming into pm_j. This indicates that at most rc_j VMs will be migrated to pm_j, satisfying the resource capacity constraint of pm_j. Note that a VM that stays at pm_j is considered as migrating from pm_j to itself.

Finally, note that the cost $\mu \cdot c\big(p(v_i), pm_j\big) + \lambda_i \cdot c\big(pm_j, sw_1\big)$ on edge (v_i, pm_j) indicates the total migration and communication cost for each of the l source VMs v_i, the cost $\mu \cdot c\big(p(v_i'), pm_j\big) + \lambda_i \cdot c\big(pm_j, sw_n\big)$ on edge (v_i', pm_j) indicates the total migration and communication cost for each of the l destination VMs v_i', while the costs on edges (s, v) and (pm_j, t) are 0. This means the minimum cost flow solution on Fig. 6(a) minimizes the cost of the $2l$ amount of flow on all the edges (v_i, pm_j) and (v_i', pm_j), which is the total migration and communication cost of all the $2l$ VMs. Therefore, the minimum cost flow algorithm yields the minimum cost of sending a $2l$ amount of flow from s_0 to t_0, demonstrating

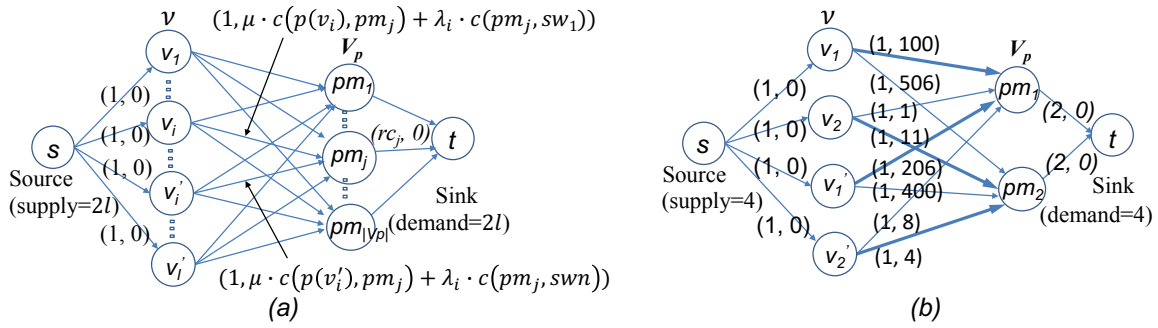


Fig. 6. (a) PAM under ordered policy is equivalent to MCF. (b) Graph transformation and MCF results (highlighted lines) for the PADC in Fig. 3(a).

that the corresponding VM migration and communication cost obtained is indeed minimum. ∎

Time Complexity. As the number of nodes, edges, and maximum edge capacity in $G'(V', E')$ are $\bar{m} \cdot |V_p|$, $\bar{m} \cdot |V_p|^2$, and $\bar{m}$ respectively, the MCF takes $O\big(\bar{m}^3 \cdot |V_p|^4 \cdot \log(\bar{m}^2 \cdot |V_p|)\big)$.

EXAMPLE 4: Fig 6(b) illustrates how the above transformation and MCF work for the same PADC in Fig. 3(a). MCF migrates v_1 and v_1' to pm_1, and v_2 and v_2' to pm_2, with total cost of 100+11+206+4+101=422. Here, 101 is the total communication cost between ingress switch sw_1 and egress switch sw_2. This migration reduces the total cost of 606 before migration by 30%. Note that as v_1 is initially located at pm_1 and v_2' at pm_2, only v_1' and v_2 actually migrate. □

3) State-of-the-Art VM Migration Scheme: Cui et al. [16] proposed a policy-aware VM management scheme named PLAN. The core concept of PLAN is *utility* of a VM migration. It is defined as the reduction in a VM's communication cost due to migration minus its migration cost. The goal of PLAN is to find a migration scheme that maximizes the *total utility* of migrating all the VMs. PLAN is a greedy algorithm that works in rounds. In each round, it determines which VM is migrated to which PM, considering the available resources, so that the utility of this migration is maximized among all VMs that have not been migrated. This process continues until all the VMs are migrated, or no further VM migration yields any positive utility. For the two VM pairs in Fig. 3(a), as there are no available resource slots, the migration cannot take place for PLAN. PLAN, however, is a heuristic algorithm that does not offer any performance guarantee. We prove in Lemma 3 that maximizing the total utilities is equivalent to minimizing total communication and migration cost in PAM, thus we can compare our algorithms with PLAN.

Lemma 3: Minimizing the total cost $C_t(m)$ in PAM is equivalent to maximizing the total utility in PLAN.

Proof: Denote the utility of migrating VM v as $u(v)$. Under migration function m, the *utility of migrating* v_i from its current PM $p(v_i)$ to another PM $m(v_i)$ is the reduction of its communication cost to the ingress switch minus the incurred migration cost. Thus $u(v_i) = \lambda_i \cdot \big(c\big(p(v_i), sw_1\big) - c\big(m(v_i), sw_1\big)\big) - \mu \cdot c\big(p(v_i), m(v_i)\big)$. Similarly, $u(v_i') = \lambda_i \cdot \big(c\big(p(v_i'), sw_n\big) - c\big(m(v_i'), sw_n\big)\big) - \mu \cdot c\big(p(v_i'), m(v_i')\big)$.

Given a p and a $\overrightarrow{\lambda}$, the total communication cost of the VMs $C_c(p)$ can be computed using Eq. 1. Thus minimizing $C_t(m)$ is equivalent to maximizing $C_c(p) - C_t(m) \overset{\text{Eqs. 1,5}}{=} \sum_{i=1}^{l} \lambda_i \cdot \Big(c\big(p(v_i), sw_1\big) + c\big(sw_n, p(v_i')\big) - c\big(m(v_i), sw_1\big) - c\big(sw_n, m(v_i')\big)\Big) - \mu \cdot \sum_{v \in \mathcal{V}} c\big(p(v), m(v)\big) = \sum_{i=1}^{l} \big(u(v_i) + u(v_i')\big)$, which is the *total utility* of migrating all the VMs. ∎

B. Unordered Policy.

1) Problem Formulation: Under unordered policy, besides a VM *migration function* $m : \mathcal{V} \to V_p$, it defines for each VM pair (v_i, v_i') an MB *traversal function* $\pi^i : [1, 2, ..., n] \to [1, 2, ..., n]$. π^i is a permutation function indicating that after VM migration, the j^{th} MB that (v_i, v_i') visits is $mb_{\pi^i(j)}$. Let $\overrightarrow{\pi} = \langle \pi^1, \pi^2, ..., \pi^l \rangle$ and let $C_t(m, \overrightarrow{\pi})$ denote the *total cost* of all the VM pairs with m and $\overrightarrow{\pi}$. Then $C_t(m, \overrightarrow{\pi}) =$

$$\sum_{i=1}^{l} \Big(\mu \cdot c\big(p(v_i), m(v_i)\big) + \mu \cdot c\big(p(v_i'), m(v_i')\big)\Big) + \sum_{i=1}^{l} \lambda_i \cdot \Bigg(\sum_{j=1}^{n-1} c\Big(sw\big(\pi^i(j)\big), sw\big(\pi^i(j+1)\big)\Big) + c\Big(m(v_i), sw\big(\pi^i(1)\big)\Big) + c\Big(sw\big(\pi^i(n)\big), m(v_i')\Big)\Bigg). \tag{6}$$

The first and second terms in Eq. 6 are the total migration cost and total communication cost respectively. The objective of PAM under unordered policy is to find an m and a $\overrightarrow{\pi}$ to minimize $C_t(m, \overrightarrow{\pi})$ while satisfying resource constraints of PMs: $|\{v \in \mathcal{V} | m(v) = i\}| \leq rc_i, \forall i \in V_p$.

2) VM Migration Algorithm for Unordered Policy: Algo. 4 below first computes costs for all the $|V_p| \cdot (|V_p| + 1)/2$ policy routes (lines 2-6). Then, for each VM pair (in the non-ascending order of their traffic rates), it finds a PM pair to migrate to, such that the resulting cost for this VM pair is the minimum among all the unassigned VM pairs in this round (lines 7-22). After the entire migration scheme m is computed, it finally migrates the VMs and returns the total cost (lines 23 and 24). Its takes $O(|V_p|^2 \cdot (n^3 + l))$.

Algorithm 4: PAM Algorithm for Unordered Policy.
Input: A PADC with unordered policy $\{mb_1, mb_2, ..., mb_n\}$, $V_p = \{pm_i\}$, resource capacity rc_i, VM pair placement p.
Output: A migration scheme m and the total cost $C_t(m, \overrightarrow{\pi})$.

Notations: i, j: indices for PM pairs; k: index for VM pairs.
$c_{i,j}$: the total cost of a VM pair if its source VM is migrated to $pm(i)$ and destination VM to $pm(j)$.
a, b: indices of a PM pair that gives the minimum total cost.
$avail(pm_i)$: number of available slots at pm_i, initially rc_i.

1. $m = \phi$, $C_t(m, \overrightarrow{\pi}) = 0$, $k = 1$, $\lambda_1 \geq \lambda_2 ... \geq \lambda_l$;
2. **for** $(i = 1; i \leq |V_p|; i{+}{+})$
3. **for** $(j = i; j \leq |V_p|; j{+}{+})$
4. Compute $pr(i, j)$ using Algo. 2;
5. **end for;**
6. **end for;**
7. **while** $(k \leq l)$ // find PM pair for VM pair (v_k, v_k')
8. $c_{min} = \infty$; // minimum total cost for (v_k, v_k')
9. **for** $(i = 1; i \leq |V_p|; i{+}{+})$
10. **if** $(avail(pm_i) == 0)$ **continue**; // pm_i is full
11. **for** $(j = i; j \leq |V_p|; j{+}{+})$
12. **if** $\big((avail(pm_j) == 0) \vee (i == j \wedge avail(pm_j) \leq 1)\big)$ **continue**; // pm_j is full
13. $c_{i,j} = 0$;
14. $c(pm_i) = \mu \cdot c(p(v_k), pm_i)$, // cost of migrating v_k
$c(pm_j) = \mu \cdot c(p(v_k'), pm_j)$; // to pm_i, v_k' to pm_j
15. $c_{i,j} = \lambda_k \cdot pr(i, j) + c(pm_i) + c(pm_j)$;
16. **if** $(c_{i,j} < c_{min})$ $a = i$, $b = j$, $c_{min} = c_{i,j}$;
17. **end for;**
18. **end for;**
19. $m = m \cup \{(pm_a, pm_b)\}$; // update migration scheme m
20. $C_t(m, \overrightarrow{\pi})$ += c_{min}; // update total cost
21. $avail(pm_a)--$, $avail(pm_b)--$;
22. **end while;**
23. Migrate $(v_1, v_1{}')$, ..., $(v_l, v_l{}')$ according to m;
24. **RETURN** m and $C_t(m, \overrightarrow{\pi})$.

EXAMPLE 5: For the two VM pairs stored in the PADC of Fig. 3 (a), Algo. 4 will migrate both v_1 and v_1' to pm_1, resulting in cost of 406 for this pair. As pm_1 is now full, it will then migrate both v_2 and v_2' to pm_2, resulting in a cost of 16 for this pair. The total cost of the two pairs is 422. □

Theorem 5: Under unordered policy, PAM is NP-hard even for one pair of VMs (i.e., $l = 1$).

Proof: We reduce a variation of the *s-t traveling salesman path problem* to this special case. By variation, we mean that in a complete graph $K = (V_K, E_K)$, each node in V_K has a cost. Thus, the cost of the *s-t* shortest Hamiltonian path includes the costs of s and t. The rest of the proof is then similar to that in Theorem 2 with one augmentation: For pm_i, its cost $c(pm_i)$ is the migration cost of v_1 from $p(v_1)$ to pm_i; for pm_j, its cost $c(pm_j)$ is the migration cost of $v_1{}'$ from $p(v_1')$ to pm_j. ∎

Theorem 6: Algo. 4 achieves 2-approximation when $l = 1$.

Proof: For Algo. 4, let the PM pair that (v_1, v_1') migrate to be (pm_a, pm_b). Let their optimal VM migration be $(pm_{a'}, pm_{b'})$ and their optimal total cost be $C_t^{opt}(m, \overrightarrow{\pi})$. The total cost of (v_1, v_1') computed by Algo. 4 $C_t(m, \overrightarrow{\pi}) = \lambda_1 \cdot pr(a, b) + \mu \cdot c(p(v_1), pm_a) + \mu \cdot c(p(v_1'), pm_b) \leq \lambda_1 \cdot pr(a', b') + \mu \cdot c(p(v_1), pm_{a'}) + \mu \cdot c(p(v_1'), pm_{b'}) \leq 2 \cdot \lambda_1 \cdot opr(a', b') + 2 \cdot \mu \cdot c(p(v_1), pm_{a'}) + 2 \cdot \mu \cdot c(p(v_1'), pm_{b'}) = 2 \cdot C_t^{opt}(m, \overrightarrow{\pi})$. ∎

VI. Heterogeneous Resource Demands

We denote the corresponding PAL and PAM problems with heterogeneous VM resource demands as PAL-H and PAM-H, wherein a VM $v \in \mathcal{V} = \{v_1, v_1', v_2, v_2', ..., v_l, v_l'\}$ consumes d_v resource slots. Thus, it must be that $\sum_{i=1}^{|V_p|} rc_i \geq \sum_{v \in \mathcal{V}} d_v$ to ensure enough PM resources to place all the VMs. Recall that the resource capacity of PM pm_i is rc_i.

A. PAL-H.

The objective of PAL-H is to find a VM placement p to minimize the total VM communication cost $C_c(p)$, shown in Eq 1, while satisfying the resource constraint of PMs: $\sum_{v \in \mathcal{V} \wedge p(v) = i} d_v \leq rc_i, \forall i \in V_p$. Although PAL can be solved optimally and efficiently, we show that PAL-H is NP hard.

Theorem 7: PAL-H is NP-hard.

Proof: We show that the bin-packing with variable sizes and costs problem (VBPP) [35] is a special case of PAL-H. Given n items $I = \{1, 2, ..., n\}$, each with a size s_i $(1 \leq i \leq n)$, and infinite number of bins B, where bin $j \in B$ has a capacity C_j and cost c_j, the goal of the VBPP is to store the n items into a set of bins $\{B_1, B_2, ..., B_m\} \subset B$ in order to minimize the total cost of the bins $\sum_{j=1}^{m} c_j$ while satisfying $\sum_{i \in B_j} s_i \leq C_j$. VBPP generalizes the classic bin-packing problem (BPP) [14], wherein all bins have the same capacities and costs, and its goal is to minimize the total number of bins used.

PAL-H, wherein the VMs are the items and the PMs are the bins, generalizes VBPP in two ways. First, in addition to its resource demand, each VM has a traffic rate (recall that the cost of placing a VM into a PM is the PM's communication cost multiplied by the VM's traffic rate). Second, as VM communication is pair-wise, PAL-H needs to place two items (i.e., the source and destination VMs) into the bins (i.e., the PMs) simultaneously. As such, PAL-H needs to address the potential conflict of placing two VMs in the same PM. We refer to this as *VM placement conflict*. When all the VMs have the same traffic rates, and there is no VM placement conflict, PAL-H becomes the VBPP. As VBPP is NP-hard [35], PAL-H is also NP-hard. ∎

Due to the heterogeneous VM resource demands, the resource capacities of PMs become an important factor in designing PAL-H algorithms. We give the definitions below.

Definition 3: (**Cost-Capacity Ratios of PMs.**) A PM pm's *ingress cost-capacity ratio*, denoted as $c_{in}(pm)$, is the ratio between the pm's ingress cost (i.e., the cost between pm and the ingress switch sw_1) and resource capacity; i.e., $c_{in}(pm) = c\big(pm, sw_1\big)/rc_{pm}$. A PM pm's *egress cost-capacity ratio* $c_e(pm)$ is the ratio between its egress cost (i.e., the cost between pm and the egress switch sw_n) and its capacity; i.e., $c_e(pm) = c\big(pm, sw_n\big)/rc_{pm}$. □

As each PM has both a cost and a resource capacity, the *best* PMs for VM placement that achieves minimum VM communication cost are those with the smallest costs and largest capacities. Thus, PMs with small cost-capacity ratios are desired for effective VM placement.

Definition 4: (**Optimal Ingress/Egress PM Set.**) An *ingress PM Set* (i.e., IPS) is a set of source PMs that store

the l source VMs. An *egress PM Set* (i.e., EPS) is a set of destination PMs that store the l destination VMs. The cost of an IPS (and EPS) is the sum of the ingress (and egress) costs of all its PMs. A pair of IPS and EPS is *optimal*, denoted as $(\text{IPS}^o, \text{EPS}^o)$, if the sum of their costs is the minimum among all pairs of IPS and EPS. □

Note that while the IRS and ERS defined in Section IV for PAL are mutually disjoint sets, IPS and EPS are not. While a source slot in a PM in PAL can only store one VM of unit sizes, a PM in PAL-H can store multiple VMs of varying sizes.

Definition 5: (**Greedy Ingress/Egress PM Set**) A *greedy ingress* PM Set, denoted as GIPS, is the first a PMs in non-descending order of their ingress cost-capacity ratio that can store the l pairs of VMs. A *greedy egress* PM Set GEPS is the first b PMs in non-descending order of their egress cost-capacity ratio that can store the l pairs of VMs. □

Lemma 4: GIPS contains at least one IPS^o, and GEPS contains at least one EPS^o.

Proof: We only prove the case of GIPS and IPS^o, and the other can be done similarly. For the l pairs of VMs $\mathcal{V} = \{v_1, v'_1, v_2, v'_2, ..., v_l, v'_l\}$, let $D_v = \sum_{i=1}^{l} d_{v_i}$ and $D'_v = \sum_{i=1}^{l} d_{v'_i}$ be the total resource demands of the source and destination VMs, respectively. Thus, the total resource capacity of GIPS $\sum_{i=1}^{a} rc_i \geq \sum_{v \in \mathcal{V}} d_v = D_v + D_{v'}$. This means that after allocating D_v amount of resources for the source VMs, there is still at least $D_{v'}$ amount left that can be allocated for destination VMs. Thus, GIPS contains at least one IPS.

Next, we prove that at least one IPS contained in GIPS is IPS^o. By way of contradiction, let's assume that none of them is an IPS^o. This means that at least one PM in IPS^o, say pm_a, is not one of the top a PMs with the smallest ingress cost-capacity ratio. Given that the GIPS has sufficient resources for one IPS and one EPS, we can identify a PM in the IPS, denoted as pm_b, that is not in IPS^o, with $rc_{pm_b} > rc_{pm_a}$ and $c_{in}(pm_b) < c_{in}(pm_a)$. Therefore, we can get another IPS, which is $\text{IPS}^o - \{pm_a\} + \{pm_b\}$, that has a total ingress cost less than that of IPS^o, contradicting that IPS^o is optimal. ∎

Two PAL-H Algorithms. Next, we propose two efficient algorithms, viz. Algo. 5 and Algo. 6, to solve PAL-H.

Next-Fit Algorithm. Algo. 5 generalizes the well-known NextFit algorithm [15] in the classic bin-packing problem by considering the cost-capacity ratios of PMs, the traffic rates of the VM pairs, and the placement conflict of the VMs. It first identifies the GIPS list $\mathcal{I}$ and GEPS list $\mathcal{E}$ and sorts VM pairs in a non-increasing order of their traffic rates (lines 1-3). Then, to place each VM pair v_k and v'_k, it finds the next PMs on $\mathcal{I}$ and $\mathcal{E}$ that have enough resources (lines 5 and 6), and stores the PM IDs in I^{opt} and E^{opt}, respectively. If these two PMs are different (lines 7-9), we place v_k and v'_k and move to the next VM pair. Otherwise, it encounters a placement conflict, which attempts to place v_k and v'_k into the same PM $\mathcal{I}[i]$. To sovle the placement conflict, there are three cases: (a) PM $\mathcal{I}[i]$ has enough resources to place both v_k and v'_k (lines 11-13), (b) PM $\mathcal{I}[i]$ only has enough resources for v_k, while v'_k must be placed at next PM $\mathcal{E}[j]$ with sufficient resources (lines 14-17), and (c) PM $\mathcal{I}[i]$ only has enough resource for v'_k, while v_k is placed at next $\mathcal{I}[i]$ with sufficient resources (lines 18-21). Finally, it places the source VMs into the PMs in I^{opt} and the destination VMs into the PMs in E^{opt}, and returns the VM placement p and the total communication cost $C_c(p)$ (lines 25-31).

Note that, unlike Algo. 1 for PAL, which can sort VM pairs in non-ascending order of their traffic rates after source and destination source slots are found (line 23 in Algo. 1), sorting VM pairs in Algo. 5 (line 3) must be done before finding the source and destination PMs, as different VMs require different resource capacities.

Finding GIPS and GEPS takes $O(|V_p| \cdot \lg(|V_p|))$, where $|V_p|$ is the number of PMs. Finding I^{opt} and E^{opt} takes l rounds, each could take $O(|V_p|)$. Sorting all the VM pairs takes $O(l \cdot \lg l)$ and calculating $C_c(p)$ takes $O(l)$. As l is bounded by $(|V_p| \cdot \bar{m})$, the time complexity of Algo. 5 is $O(|V_p|^2 \cdot \bar{m})$, which is more efficient than Algo. 1 for PAL.

Algorithm 5: NextFit VM placement for PAL-H.
Input: A PADC with ordered policy $(mb_1, mb_2, ..., mb_n)$, VM pairs $\mathcal{P}$, resource demand of VM v is d_v, $V_p = \{pm_i\}$, resource capacity of pm_i is rc_i.
Output: A VM placement p and the total communication cost of all VM pair $C_c(p)$.
Notations: $\mathcal{I}$ and $\mathcal{E}$: arrays of GIPS and GEPS.
I^{opt}, E^{opt}: the final source and destination PMs.
i, j: indices for PM IDs in $\mathcal{I}$ and $\mathcal{E}$ respectively.
k: indices for PM IDs in I^{opt} and E^{opt}, and VM pairs.
$rc(pm_i)$: current resource capacity of pm_i, initially rc_{pm_i}.
M^s: the set of source PMs allocated so far, initially ϕ.
M^d: the set of destination PMs allocated so far, initially ϕ.
1. $i = j = k = 1$, $C_c(p) = 0$, $p = I^{opt} = E^{opt} = \phi$ (empty set);
2. Find GIPS (of a PMs), store the PM IDs in an array $\mathcal{I}$;
   Find GEPS (of b PMs), store the PM IDs in an array $\mathcal{E}$;
3. Sort VM pairs in non-ascending order of their traffic rates, assume $\lambda_1 \geq \lambda_2 ... \geq \lambda_l$;
4. **for** $(1 \leq k \leq l)$ // find PMs for (v_k, v'_k)
5. **while** $(rc(\mathcal{I}[i]) < d_{v_k})$ i++; // find next PM $\mathcal{I}[i]$) or $\mathcal{E}[j]$
6. **while** $(rc(\mathcal{E}[j]) < d_{v'_k})$ j++; // with enough resources
7. **if** $(\mathcal{I}[i] \neq \mathcal{E}[j]$) // v_k placed at $\mathcal{I}[i]$ and v'_k at $\mathcal{E}[j]$
8. $I^{opt}[k] = \mathcal{I}[i]$, $E^{opt}[k] = \mathcal{E}[j]$;
9. $rc(\mathcal{I}[i]) = rc(\mathcal{I}[i]) - d_{v_k}, rc(\mathcal{E}[j]) = rc(\mathcal{E}[j]) - d_{v'_k}$;
10. **else**
11. **if** $(rc(\mathcal{I}[i]) \geq d_{v_k} + d_{v'_k})$ // both placed at PM $\mathcal{I}[i]$
12. $I^{opt}[k] = E^{opt}[k] = \mathcal{I}[i]$;
13. $rc(\mathcal{I}[i]) = rc(\mathcal{I}[i]) - d_{v_k} - d_{v'_k}$;
14. **elseif** $(rc(\mathcal{I}[i]) \geq d_{v_k})$ // v_k placed at PM $\mathcal{I}[i]$
15. $I^{opt}[k] = \mathcal{I}[i]$, $rc(\mathcal{I}[i]) = rc(\mathcal{I}[i]) - d_{v_k}$;
16. **while** $(rc(\mathcal{E}[j]) < d_{v'_k})$ j++;
17. $E^{opt}[k] = \mathcal{E}[j]$, $rc(\mathcal{E}[j]) = rc(\mathcal{E}[j]) - d_{v'_k}$;
18. **elseif** $(rc(\mathcal{I}[i]) \geq d_{v'_k})$ // v'_k is placed at PM $\mathcal{I}[i]$
19. $E^{opt}[k] = \mathcal{I}[i]$, $rc(\mathcal{I}[i]) = rc(\mathcal{I}[i]) - d_{v'_k}$;
20. **while** $(rc(\mathcal{I}[i]) < d_{v_k})$ i++;
21. $I^{opt}[k] = \mathcal{I}[i]$, $rc(\mathcal{I}[i]) = rc(\mathcal{I}[i]) - d_{v_k}$;

22. **end if;**
23. **end if;**
24. **end for;**
25. **for** $(1 \le i \le l)$ // place VM pairs and calculate cost
26. Place v_i at PMs $I^{opt}[i]$ and v'_i at $E^{opt}[i]$;
27. $p = p \cup \{(I^{opt}[i], E^{opt}[i])\}$;
28. $C_c(p)$ += $\lambda_i * \big(c(I^{opt}[i], sw_1) + c(E^{opt}[i], sw_n)\big)$;
29. **end for;**
30. $C_c(p)$ += $\sum_{i=1}^{l} \lambda_i \sum_{j=1}^{n-1} c\big(sw_j, sw_{j+1}\big)$;
31. **RETURN** p and $C_c(p)$.

Theorem 8: When all the PMs have the same costs, all the VMs have the same traffic rates, and there is no placement constraint, Algo. 5 is a 2-approximation algorithm for PAL-H.

Proof: Under those conditions, PAL-H is equivalent to finding a VM placement that minimizes the total resource capacities of PMs, and the GIPS and GEPS are the PMs in non-ascending order of their resource capacities. Since there is no placement constraint for the source and destination VMs, the PAL-H can be treated as two separate problems of placing source VMs v_k into GIPS and destination VMs v'_k into GEPS, respectively. We thus only focus on the former.

Let $B = \{pm_1, pm_2, ..., pm_l\}$ be the ordered set of PMs selected by Algo. 5 and let $B^* = \{pm_1^*, pm_2^*, ..., pm_m^*\}$ be the ordered set of PMs selected by an optimal algorithm. We want to show that $\sum_{i=1}^{l} rc(pm_i) \le 2 \times \sum_{i=1}^{m} rc(pm_i^*)$. Let $c(pm_i)$ denote the total resource capacities of the VMs that are placed in pm_i; $c(pm_i) = \sum_{v \in \mathcal{V} \wedge p(v)=i} d_v$. Clearly, $c(pm_i) \le rc(pm_i)$ and $c(pm_i^*) \le rc(pm_i^*)$. Assume the resource demands of VMs and the resource capacities of PMs are normalized so that the largest PM has a resource capacity of 1. For $1 \le i \le l$, $c(pm_i) + c(pm_{i+1}) > 1$; thus, $\sum_{i=1}^{l} c(pm_i) > (l-1)/2$. We have $\sum_{i=1}^{l} rc(pm_i) \le (l-1)+1 < 2 \times \sum_{i=1}^{l} c(pm_i)+1 = 2 \times \sum_{i=1}^{m} c(pm_i^*)+1 \le 2 \times \sum_{i=1}^{m} rc(pm_i^*)+1$. □

Algorithm 6: FirstFit VM Algorithm for PAL-H.
Input: A PADC with ordered policy $(mb_1, mb_2, ..., mb_n)$, VM pairs $\mathcal{P}$, resource demand of VM v is d_v, $V_p = \{pm_i\}$, resource capacity of pm_i is rc_i.
Output: A VM placement p and the total communication cost of all VM pair $C_c(p)$.
Notations: $\mathcal{I}$ and $\mathcal{E}$: arrays of source and destination PMs.
i, j: indices for PM IDs in $\mathcal{I}$ and $\mathcal{E}$ respectively.
k: index for PM IDs in I^{opt} and E^{opt}, as well as VM pairs.
$rc(pm_i)$: current resource capacity of pm_i, initially rc_i.
M^s: the set of source PMs allocated so far, initially ϕ.
M^d: the set of destination PMs allocated so far, initially ϕ.
$placed$: if a VM can be placed into M^s or M^d, initially false.
1. $i = j = k = 1$, $C_c(p) = 1$, $p = I^{opt} = E^{opt} = \phi$ (empty set);
2. Find GIPS (of a PMs), store their IDs in an array $\mathcal{I}$;
   Find GEPS (of b PMs), store their IDs in an array $\mathcal{E}$;
3. Sort VM pairs in non-ascending order of their traffic rates, assume $\lambda_1 \ge \lambda_2 ... \ge \lambda_l$;
4. **for** $(1 \le k \le l)$ // find PMs to store (v_k, v'_k)
5. placed = false;
6. **for** (each PM pm in M^s)
7. **if** $(d_{v_k} \le rc(pm))$
8. $I^{opt}[k] = pm$, $rc(pm) = rc(pm) - d_{v_k}$;
9. placed = true, break;
10. **end if;**
11. **end for;**
12. **if** (placed == false)
13. $I^{opt}[k] = \mathcal{I}[i]$, $rc(\mathcal{I}[i]) = rc(\mathcal{I}[i]) - d_{v_k}$;
14. $M^s = M^s \cup \{\mathcal{I}[i]\}$;
15. i++;
16. **end if;**
17. placed = false;
18. **for** (each PM pm in M^d)
19. **if** $(d_{v'_k} \le rc(pm))$
20. $E^{opt}[k] = pm$, $rc(pm) = rc(pm) - d_{v'_k}$;
21. placed = true, break;
22. **end if;**
23. **end for;**
24. **if** (placed == false)
25. $E^{opt}[k] = \mathcal{E}[j]$, $rc(\mathcal{E}[j]) = rc(\mathcal{E}[j]) - d_{v'_k}$;
26. $M^s = M^s \cup \{\mathcal{E}[j]\}$;
27. j++;
28. **end if;**
29. **end for;**
30. **for** $(1 \le i \le l)$ // place VM pairs and calculate cost
31. Place v_i at PMs $I^{opt}[i]$ and v'_i at $E^{opt}[i]$;
32. $p = p \cup \{(I^{opt}[i], E^{opt}[i])\}$;
33. $C_c(p)$ += $\lambda_i * \big(c(I^{opt}[i], sw_1) + c(E^{opt}[i], sw_n)\big)$;
34. **end for;**
35. $C_c(p)$ += $\sum_{i=1}^{l} \lambda_i \sum_{j=1}^{n-1} c\big(sw_j, sw_{j+1}\big)$;
36. **RETURN** p and $C_c(p)$.

First-Fit Algorithm. One improvement upon Algo. 5 is to place the VMs into an available PM with enough resource capacity, before allocating a new PM, in the spirit of the NextFit algorithm [15] for the bin-packing problem. When placing a source VM v_k, it places it into the first PM in the GIPS that has sufficient resource capacity and has already been placed with some VMs (lines 5 to 11). If no such PMs exist, it will then allocate a new PM from the GIPS list to place the VMs (lines 12 and 16). The destination VM v'_k is placed into the PMs in the GEPS list in a similar manner (lines 17-28). The time complexity of Algo. 6 is $O(|V_p|^2 \cdot \bar{m})$. Although we are not able to prove that it achieves a similar approximation ratio, we show via extensive simulations that it outperforms Algo. 5 consistently.

B. PAM-H.

The objective of PAM-H is to find a VM migration scheme m that minimizes total VM migration and communication cost $C_t(m)$, shown in Eqn. 5, while satisfying resource constraint of PMs: $\sum_{v \in \mathcal{V} \wedge m(v)=i} d_v \le rc_i, \forall i \in V_p$. While Section V shows that PAM is equivalent to an MCF, which can be solved

optimally and efficiently, PAM-H is indeed APX-hard.

Theorem 9: PAM-H is APX-hard.

Proof: We show that PAM-H is equivalent to an energy-efficient data redistribution problem (DRP) in sensor networks without a basestation [47], which is APX-hard. As there is no basestation to store the sensory data, DRP redistributes the data packets generated from storage-depleted *data nodes* to *storage nodes* with storage capacities for preservation. The goal of the DRP is to minimize the total energy consumption in this process. PAM-H corresponds to DRP with the same set of problem variables and objective characteristics.

First, packets in DRP have different sizes, just as each VM has a different resource demand. Meanwhile, storage nodes in DRP have storage capacities, just like PMs have resource capacities. Second, in DRP, redistributing each packet to different storage nodes costs a different amount of energy, just like migrating each VM to different PMs incurs different migration costs. Finally, the objective of minimizing the total energy consumption of data redistribution in DRP is precisely the same as the objective of minimizing the total migration and communication cost of the VMs in the PAM-H.

As such, PAM-H is equivalent to the DRP. As DRP is APX-hard [47], PAM-H is also APX-hard. ∎

As PAM-H is APX-hard, designing a polynomial-time approximation algorithm for PAM-H is unlikely. Below, we present an optimal ILP solution and an efficient heuristic algorithm for solving PAM-H.

An Optimal ILP Solution. For each source VM v_i, let $t_{i,j} = \mu \cdot c\big(p(v_i), pm_j\big) + \lambda_i \cdot c\big(pm_j, sw_1\big)$, which indicates the sum of v_i's migration cost and its communication cost to the ingress switch. For each destination VM v_i', let $t_{i,j} = \mu \cdot c\big(p(v_i'), pm_j\big) + \lambda_i \cdot c\big(pm_j, sw_n\big)$, which indicates the sum of v_i''s migration cost and its communication cost to the egress switch. $x_{v,j}$ indicate if VM v, which coulld be either v_i or v_i', is migrated to PM pm_j or not.

$$(A) \qquad \min \sum_{v \in \mathcal{V}} \sum_{pm_j \in V_p} t_{v,j} \cdot x_{v,j} \tag{7}$$

s.t.

$$x_{v,j} \in \{0,1\}, \quad \forall v \in \mathcal{V}, pm_j \in V_p \tag{8}$$

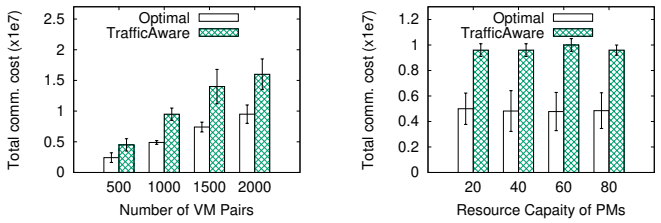
$$\sum_{pm_j \in V_p} x_{v,j} = 1, \quad \forall v \in \mathcal{V} \tag{9}$$

$$\sum_{v \in \mathcal{V}} d_v \cdot x_{v,j} \leq rc_j, \quad \forall pm_j \in V_p \tag{10}$$

Objective 7 aims to minimize the total migration and communication cost of all the VMs. Constraint 9 indicates each VM v will be migrated to a PM, and Constraint 10 indicates the resource constraint of each PM.

A Greedy Algorithm. We introduce the following definition.

Definition 6: (**Utility Resource Ratio (URR).**) The URR of migrating a VM v is defined as the ratio between the utility $u(v)$ of migrating v and the v's resource demand d_v. Under migration function m, the *utility of migrating* v_i from its current PM $p(v_i)$ to another PM $m(v_i)$ is the reduction of its communication cost minus the incurred migration cost. Thus $u(v_i) = \lambda_i \cdot \big(c\big(p(v_i), sw_1\big) - c\big(m(v_i), sw_1\big)\big) - \mu \cdot c\big(p(v_i), m(v_i)\big)$. Similarly, $u(v_i') = \lambda_i \cdot \big(c\big(p(v_i'), sw_n\big) - c\big(m(v_i'), sw_n\big)\big) - \mu \cdot c\big(p(v_i'), m(v_i')\big)$. □



(a) Varying l. $n = 3$, $rc = 40$. (b) Varying rc. $l = 1000$, $n = 3$.

Fig. 7. Comparing with TrafficAware in ordered policy, $k = 16$.

The greedy algorithm takes place in rounds. In each round, it moves (i.e., migrates) a VM to a PM that yields the largest URR. If there are multiple migrations with the same largest URR values, we will choose the one with the same PM. This process continues until all the VMs have been migrated. With the costs between any pair of PMs (or switches) precomputed, deciding which VM is migrated to which PM yields the largest URR in each round takes $O(l \cdot |V_p|)$. As there are l pairs of VMs, and l is bounded by $|V_p| \cdot \bar{m}$, the time complexity is thus $O(|V_p|^3 \cdot \bar{m}^2)$.

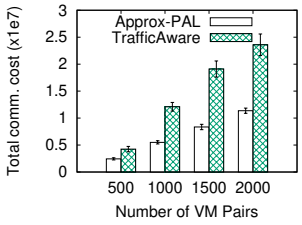
VII. Performance Evaluation

In this section, we compare our algorithms with existing work (Table II). For PAL, we name the optimal algorithm for ordered policies (Algo. 1) as **Optimal** and the approximation algorithm for unordered (Algo. 3) as **Approx-PAL**, and compare them with **TrafficAware** [41], a seminal but policy-oblivious VM placement algorithm. For PAM, we refer to the minimum cost flow-based algorithm for ordered policy as **MCF** and the approximation algorithm for unordered (Algo. 4) as **Approx-PAM**, and compare them with **PLAN** [16].

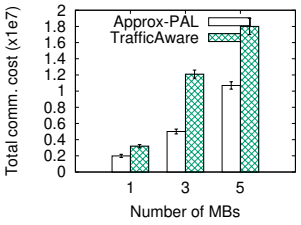
We consider fat-tree PADCs of size $k = 8$ with 128 PMs and size $k = 16$ with 1024 PMs. The traffic rates of VM pairs are in the range of [0, 1000] – Following flow characteristics found in Facebook data centers [43], 25% of VM pairs have light traffic rates in [0, 300), 70% medium traffic rates in [300, 700], and 5% heavy rates in (700,1000]. As suggested by Cisco design guide [3], we install a number of MBs on aggregation switches in the PADC. As 80% of cloud data center traffic originated by servers stays within the rack [10], for the initial VM placement in PAM, we place 80% of the VM pairs into the PMs under the same edge switches, while the remaining

TABLE II
Comparing PAM and PAL Algorithms.

	Ordered Policy	Unordered Policy	Existing Work
PAL	Optimal	Approx-PAL	TrafficAware [41]
PAM	MCF	Approx-PAM	PLAN [16]

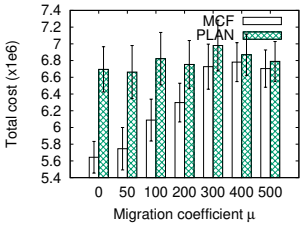


(a) Varying l. $n = 3$, $rc = 40$.

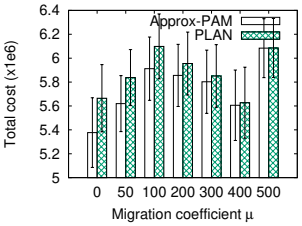


(b) Varying n. $l = 1000$, $rc = 40$.

Fig. 8. Comparing with TrafficAware in unordered policy, $k = 16$.



(a) Ordered Policy.



(b) Unordered Policy.

Fig. 9. Comparing with PLAN, $k = 8$, $l = 1000$, $n = 3$, $rc = 40$.

20% under different edge switches. Each data point represents the average of 20 runs, along with a 95% confidence interval. In each run, a new set of VM pairs is placed (for PAL) or migrated (for PAM) in the PADC. The codes are available at https://github.com/vtran42/pal-pam-datacenter.

Comparing with TrafficAware. As TrafficAware only assigns VMs to the same PMs or PMs in close proximity, and does not consider the proximity of the PMs to the MBs, we implement TrafficAware as follows for fair comparison. In ordered policy, it places VM pairs (in non-ascending order of their traffic rates) to the PMs that are closest to the ingress switch. In unordered policy, it works similarly to Algo. 3 but only considers the Hamiltonian cycle case, as TrafficAware always places VM pairs in the same PM if possible.

For ordered policy, Fig. 7(a) varies the number of VM pairs l and shows that Optimal yields 46-49% less costs than TrafficAware. Fig. 7(b) varies resource capacities of PMs rc and shows that Optimal outperforms TrafficAware by around 48%. Fig. 8 compares Approx-PAL and TrafficAware under an unordered policy. It varies l as well as the number of MBs n and shows that Approx-PAL outperforms TrafficAware by 37-58% in all scenarios. The above results are evident as Optimal is optimal and Approx-PAL is a 2-approximation policy-aware algorithm, while TrafficAware is policy-oblivious, inducing enormous traffic when VM communication traverses the MBs.

Effects of VM Migrations. Fig. 10 investigates how much cost reduction VM migration brings to a PADC ($k = 8$, $l = 1000$, $n = 3$) compared to without migration. VM migrations take place in *epochs*. At the beginning of each epoch, VM pairs adjust their traffic rates to new values within the range [0, 1000] in accordance with the aforementioned Facebook flow pattern. For migrations, it then executes MCF and calculates the total migration and communication cost. For no migration, it simply recalculates the total communication cost using the new traffic rates. We set the migration coefficient μ as 10 and 50, and let the PADC run continuously for ten epochs. Fig. 10 shows the total cost of VM pairs in each epoch with and without VM migration, for both $rc = 40$ and 80. The cost of $\mu = 50$ is larger than that of $\mu = 10$, as migration costs increase with the increase of μ. In either case, the cost of $rc = 80$ is smaller than that of $rc = 40$, as there are more resource slots available to achieve cost-efficient VM migrations. In all cases, VM migration reduces the total costs by up to 25% ($\mu = 10$ and $rc = 80$) compared to those without migration.

Comparing with PLAN. We then compare our PAM algorithms with PLAN by increasing the migration coefficient μ. As PLAN is only designed for ordered policies, for the purpose of comparison with unordered policies, we implement it as the greedy algorithm below. For each VM pair, it finds the MB closest to the source VM as the ingress MB and the one closest to the destination VM as the egress MB. It then finds an MB sequence by starting from the ingress MB, visiting the closest unvisited MB, and so on, until all the MBs are visited, and finally visiting the egress MB. Fig. 9(a) shows that under the ordered policy, the MCF outperforms the PLAN by around 20% when μ is small. With the increase of μ, PLAN and MCF start to perform closely due to the high migration cost. Fig. 9(b) shows that under an unordered policy, Approx-PAM outperforms PLAN slightly for the entire range of μ.

Finally, Fig. 11 compares Approx-PAM and PLAN by varying rc while fixing μ as 20, and shows that Approx-PAM outperforms PLAN for the entire range of rc. With the increase in the rc, the performance difference between Approx-PAM and Greedy becomes larger, around 30%.

Heterogeneous VM Resource Demands. Finally, we consider that VMs have different resource demands, and compare various algorithms solving PAL-H and PAM-H, respectively. In all simulations below, the resource demands of the VMs are randomly generated in the range of [1, 8]. We refer to the PMs that are allocated to place the VMs as the *active PMs*. The code for PAL-H and PAM-H is available at https://github.com/chrisagonza97/FatTreePython/.

PAL-H. Fig. 12 compares two PAL-H algorithms viz. Algo. 5 (i.e., **Next-Fit**) and Algo. 6 (i.e., **First-Fit**). Fig. 12(a) shows that under a wide range of numbers of VM pairs, First-Fit

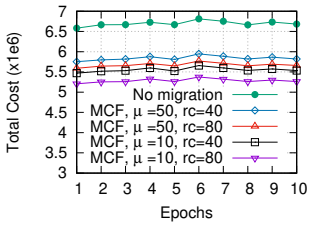


Fig. 10. Effects of VM migration.

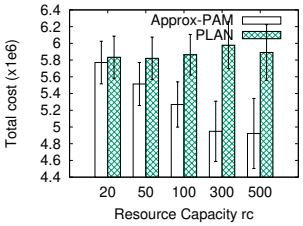
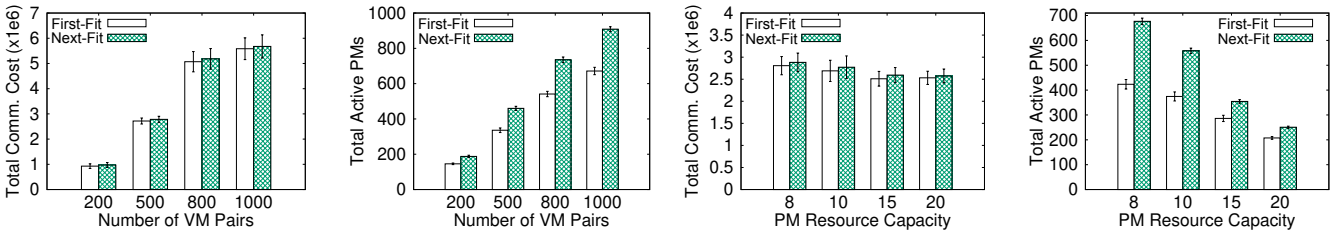


Fig. 11. Comparing with PLAN.



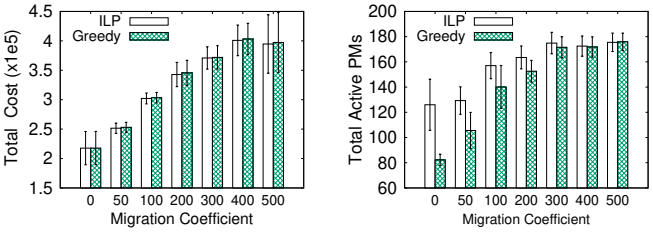
(a) Varying l. $rc = 12$. (b) Varying l. $rc = 12$. (c) Varying rc. $l = 500$. (d) Varying rc. $l = 500$.

Fig. 12. Comparing First-Fit and Next-Fit of PAL-H, $k = 16$, $n = 3$.

constantly yields a slightly smaller total VM communication cost than that of Next-Fit, as it always tries to utilize the existing PMs before allocating a new one. This is further validated by Fig. 12(b), which shows that First-Fit uses fewer active PMs for VM placement, with up to 22.2% fewer active PMs than Next-Fit. Note that although $rc = 10$ would be theoretically enough to ensure sufficient PM resources to place 1000 pairs of VMs, as $\sum_{i=1}^{|V_p|} rc_i \geq \sum_{v \in \mathcal{V}} d_v$, practically, it is not enough due to the PM resource fragmentation during the VM placement process. We thus have to set $rc = 12$. Fig. 12(c) and (d) show that when increasing the resource capacities of the PMs, the total communication cost of VMs and the number of active PMs decrease for both algorithms. However, First-Fit still yields a lower VM communication cost with fewer active PMs, as it prioritizes placing VMs in PMs with smaller costs before allocating PMs with larger costs.

PAM-H. VMs are initially placed randomly on the PMs. Then, their traffic rates are randomized while still adhering to the Facebook flow characteristics mentioned earlier. After that, the two PAM-H algorithms, viz. **ILP** and **Greedy** are applied to the same instance. As ILP takes time to compute, we use small fat-tree networks of $k = 8$ with $l = 100$ VM pairs. Fig. 13 compares them by varying the migration coefficient μ. Fig. 13(a) shows that Greedy performs very closely to the ILP in terms of total VM costs, constantly within 95.4% of the optimal cost, although the variance of the data gets larger with the increase of μ for both ILP and Greedy. More interestingly, Fig. 13(b) shows that the Greedy constantly yields fewer active PMs than the ILP. This could save a large amount of energy for the Greedy, as those inactive PMs can be turned off. As ILP focuses on minimizing total costs, it is achieved at the expense of using as many active PMs as possible.

Finally, it is evident that the performance difference between ILP and Greedy is more pronounced at smaller values of μ,



(a) Varying l. $rc = 12$. (b) Varying l. $rc = 12$.

Fig. 13. Comparing ILP and Greedy of PAM-H. $k = 8$, $n = 3$, $l = 100$, $rc = 12$.

with Greedy utilizing approximately 80 active PMs, while ILP uses around 124 on average. This is attributed to the fact that when μ is small, which means migration does not incur much cost, the PAM-H becomes PAL-H. Due to the flattened layout of PMs in the fat-tree topology, there are many PMs with the same or similar costs to the ingress or egress switch. When ILP places VMs into the PMs, it could choose arbitrary PMs as long as they provide the smallest costs. In contrast, for the Greedy algorithm, when multiple migrations achieve the same minimum cost (i.e., the largest URR values), it chooses the one with the same PM, thereby dramatically reducing the number of active PMs.

VIII. Conclusions and Future Work

In this paper, we focus on the dynamic traffic in cloud data centers and study PAL and PAM: VM placement and VM migration in the policy-aware data centers, where VM traffic must visit a sequence of MBs for various security and performance purposes. We demonstrated that VM migration is an effective technique for alleviating dynamic VM traffic in PADCs, thereby saving cloud resources. Working together, PAL and PAM place and then migrate VMs in the event of dynamic traffic fluctuation, achieving optimal and near-optimal network resource management for a PADC's lifetime. We uncovered a suite of new policy-aware problems and designed optimal, approximation, and heuristic algorithms. We also demonstrated that our results outperform those of various state-of-the-art methods.

Our further work has two directions. First, we will consider the PADC with virtualized MB and VNFs, rather than hardware MBs. As VNFs can be placed and migrated as software, designing a holistic VNF+VM migration scheme to achieve optimal resource utilization in dynamic PADCs becomes a new challenging problem. Second, as reinforcement learning has come to play a significant role in cloud data center traffic engineering and management [19], we will design reinforcement learning-based VM placement and migration to further optimize cloud resource usage in dynamic policy-aware data centers.

Acknowledgment

This work was supported by NSF Grant CNS-1911191.

References

[1] Amazon lex. https://aws.amazon.com/lex/.
[2] Christofides algorithm. https://en.wikipedia.org/wiki/Christofides_algorithm.

[3] Cisco virtualized multi-tenant data center, version 2.0 compact pod design guide. http://hyperurl.co/hpj2xt.

[4] Slack, the collaboration software that moves work forward. http://slack.com.

[5] A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[6] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, 2008.

[8] M. Alicherry and T.V. Lakshman. Optimizing data access latencies in cloud systems by intelligent virtual machine placement. In *Proc. of IEEE INFOCOM 2013*.

[9] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan. Learn-as-you-go with megh: Efficient live migration of virtual machines. *IEEE Transactions on Parallel and Distributed Systems*, 30(8):1786–1801, 2019.

[10] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC 2010*.

[11] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues, 2002. https://tools.ietf.org/html/rfc3234.

[12] F. Carpio, W. Bziuk, and A. Jukan. Scaling migrations and replications of virtual network functions based on network traffic forecasting. *Computer Networks*, 203, 2022.

[13] C. Clark, K. Fraser, and S. Hand. Live migration of virtual machines. In *NSDI 2005*.

[14] Edward G. Coffman Jr., János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. *Bin Packing Approximation Algorithms: Survey and Classification*. 2013.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[16] L. Cui, F. P. Tso, D. P. Pezaros, W. Jia, and W. Zhao. Plan: Joint policy- and network-aware vm management for cloud data centers. *IEEE Transactions on Parallel and Dis. Sys.*, 28(4):1163–1175, 2017.

[17] Y. Cui, Z. Yang, S. Xiao, X. Wang, and S. Yan. Traffic-aware virtual machine migration in topology-adaptive dcn. *IEEE/ACM Transactions on Networking*, 25(6):3427 – 3440, 2017.

[18] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca. An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. *IEEE/ACM Transactions on Networking*, 25(4):2008–2025, 2017.

[19] J. Chen et al. Fault tolerance oriented sfc optimization in sdn/nfv-enabled cloud environment based on deep reinforcement learning. *IEEE Transactions on Cloud Computing*, 12(1):200–218, 2024.

[20] J. Zeng et al. Adaptive drl-based virtual machine consolidation in energy-efficient cloud data center. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2991–3002, 2022.

[21] X. Liu et al. Joint availability enhancement and traffic optimization of virtual cluster allocation in cloud datacenters. *IEEE Transactions on Network and Service Management*, 17(3):1554–1567, 2020.

[22] Y. Chen et al. A combined trend virtual machine consolidation strategy for cloud data centers. *IEEE Transactions on Computers*, 73(9):2150–2164, 2024.

[23] Y. Laili et al. An iterative budget algorithm for dynamic virtual machine consolidation under cloud computing environment. *IEEE Transactions on Services Computing*, 14(1):30–43, 2021.

[24] Yao Lu et al. Computing in the era of large generative models: From cloud-native to ai-native, 2024. https://arxiv.org/abs/2401.12230.

[25] Z. Han et al. Energy-efficient dynamic virtual machine management in data centers. *IEEE/ACM Transactions on Networking*, 27(1):344–360, 2019.

[26] J. Fan, C. Guan, Y. Zhao, and C. Qiao. Availability-aware mapping of service function chains. In *IEEE INFOCOM 2017*.

[27] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *USENIX NSDI 2014*.

[28] H. Flores, V. Tran, and B. Tang. Pam & pal: Policy-aware virtual machine migration and placement in dynamic cloud data centers. In *IEEE INFOCOM 2020*.

[29] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, 22:1–29, 1997.

[30] A. Gushchin, A. Walid, and A. Tang. Scalable routing in sdn-enabled networks with consolidated middleboxes. In *ACM Hotmiddlebox*, 2015.

[31] J.A. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291 – 295, 1991.

[32] H. Huang, S. Guo, J. Wu, and J. Li. Service chaining for hybrid network function. *IEEE Transactions on Cloud Computing*, 7:1082–1094, 2019.

[33] Y. Jiang, Y. Cui, W. Wu, Z. Xu, J. Gu, K. K. Ramakrishnan, Y. He, and X. Qian. Speedybox: Low-latency nfv service chains with cross-nf runtime consolidation. In *IEEE ICDCS 2019*.

[34] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *ACM SIGCOMM 2008*.

[35] J. Kang and S. Park. Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, 147(2):365–372, 2003.

[36] T. Kuo, B. Liou, K. C. Lin, and M. Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions on Networking*, 26(4):1562–1576, Aug 2018.

[37] X. Li, J. Wu, S. Tang, and S. Lu. Let's stay together: Towards traffic aware virtual machine placement in data centers. In *Proc. of IEEE INFOCOM 2014*.

[38] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin. Improve service chaining performance with optimized middlebox placement. *IEEE Transactions on Services Computing*, 10(4):560–573, 2017.

[39] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu. On dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management*, 14(3):543–553, 2017.

[40] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer. Remedy: Network-aware steady state vm management for data centers. In *the NETWORKING 2012*.

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

[42] R. Mohamed, M. Avgeris, A. Leivadeas, and I. Lambadaris. Optimizing resource fragmentation in virtual network function placement using deep reinforcement learning. *IEEE Transactions on Machine Learning in Communications and Networking*, 2, 2024.

[43] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM 2015*.

[44] G. Sallam, G.R. Gupta, B. Li, and B. Ji. Shortest path and maximum flow problems under service function chaining constraints. In *IEEE INFOCOM 2018*.

[45] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *INFOCOM 2017*.

[46] Justine Marie Sherry. *Middleboxes as a Cloud Service*. Ph.D. Thesis, UC Berkeley, 2016.

[47] B. Tang, N. Jaggi, H. Wu, and R. Kurkal. Energy-efficient data redistribution in sensor networks. *ACM Trans. Sen. Netw.*, 9(2):11:1–11:28, April 2013.

[48] V. Tran, J. Sun, B. Tang, and D. Pan. Traffic-optimal virtual network function placement and migration in dynamic cloud data centers. In *IEEE IPDPS 2022*.

[49] S. Tuli, G. Casale, and N. R. Jennings. Pregan: Preemptive migration prediction network for proactive fault-tolerant edge computing. In *IEEE INFOCOM 2022*, pages 670–679.

[50] D. Wang, W. Zhang, X. Han, J. Lin, and Y. Tian. A multi-objective virtual network migration algorithm based on reinforcement learning. *IEEE Transactions on Cloud Computing*, 11(02):2039–2056, 2023.

[51] H. Wang, Y. Li, Y. Zhang, and D. Jin. Virtual machine migration planning in software-defined networks. In *Infocom 2015*.

[52] W. Wei, H. Gu, Z. Xiao, and Y. Chen. Energy efficient and multi-resource optimization for virtual machine placement by improving moea/d. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–15, 2025.

[53] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, and Y. Pan. Stochastic load balancing for virtual resource management in datacenters. *IEEE Transactions on Cloud Computing (Early Access)*, 2018.

[54] F. Zhang, G. Liu, X. Fu, and Ramin Yahyapour. A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Communications Surveys & Tutorials*, 20:1206–1243, 2018.

[55] J. Zhang, F. Ren, and C. Lin. Delay guaranteed live migration of virtual machines. In *INFOCOM 2014*.

[56] Z. Zhang, H. Cao, S. Su, and W. Li. Energy aware virtual network migration. *IEEE Transactions on Cloud Computing*, 10(2):1173–1189, 2022.