Multimedia Systems
DOI 10.1007/s00530-017-0536-y

REGULAR PAPER

Latency-aware virtual desktops optimization in distributed clouds

Tian Guo[1] · **Prashant Shenoy**[2] · **K. K. Ramakrishnan**[3] · **Vijay Gopalakrishnan**[4]

Received: 14 March 2015 / Accepted: 19 January 2017

Abstract Distributed clouds offer a choice of data center locations for providers to host their applications. In this paper, we consider distributed clouds that host virtual desktops which are then accessed by users through remote desktop protocols. Virtual desktops have different levels of latency-sensitivity, primarily determined by the actual applications running and affected by the end users' locations. In the scenario of mobile users, even switching between 3G and WiFi networks affects the latency-sensitivity. We design VMShadow, a system to automatically optimize the location and performance of latency-sensitive VMs in the cloud. VMShadow performs black-box fingerprinting of a VM's network traffic to infer the latency-sensitivity and employs both ILP and greedy heuristic based algorithms to move highly latency-sensitive VMs to cloud sites that are closer to their end users. VMShadow employs a WAN-based live migration and a new network connection migration protocol to ensure that the VM migration and subsequent changes to the VM's network address are transparent to end-users. We implement a prototype of VMShadow in a nested hypervisor and demonstrate its effectiveness for optimizing the performance of VM-based desktops in the cloud. Our experiments on a private as well as the public EC2 cloud show that VMShadow is able to discriminate between latency-sensitive and insensitive desktop VMs and judiciously moves only those that will benefit the most from the migration. For desktop VMs with video activity, VMShadow improves VNC's refresh rate by 90% by migrating virtual desktop to the closer location. Transcontinental remote desktop migrations only take about 4 min and our connection migration proxy imposes 13 µs overhead per packet.

Keywords Distributed clouds · Virtual desktop · Placement algorithm

Communicated by R. Rejaie.

A preliminary version of this work appeared at the ACM MMSys 2014 Conference [1].

✉ Tian Guo
tian@cs.wpi.edu

Prashant Shenoy
shenoy@cs.umass.edu

K. K. Ramakrishnan
kk@cs.ucr.edu

Vijay Gopalakrishnan
gvijay@research.att.com

[1] Worcester Polytechnic Institute, Worcester, USA

[2] Umass Amherst, Amherst, USA

[3] UC Riverside, Riverside, USA

[4] AT&T Labs, Bedminster, USA

1 Introduction

Hosting online applications on cloud platforms has become a popular paradigm. Applications ranging from multi-tier web applications, gaming and individual desktops are being hosted out of virtualized resources running in commercial cloud platforms or in a private cloud run by enterprises. The wide range of applications supported have diverse needs in terms of computation, network bandwidth and latency. To accommodate this and to provide geographic diversity, cloud platforms have become more distributed in recent years. Many cloud providers now offer a choice of several locations for hosting a cloud application. For instance, Amazon's EC2 cloud provides a choice

Published online: 24 March 2017

of eleven global locations across four continents. Similarly, enterprise-owned private clouds are distributed across a few large data centers as well as many smaller branch office sites. Such distributed clouds enable application providers to choose the geographic region(s) best suited to the needs of the particular application.

A concurrent trend is the growing popularity of virtual desktops (VDs) where the desktop PC of a user is encapsulated into a virtual machine (VM) and this VM is hosted on a remote server or the cloud; users then access their desktop applications and their data files via a remote desktop protocol such as VNC (and via thin clients). This trend—known as virtual desktop infrastructure (VDI)—is being adopted in the industry due to numerous benefits. First of all, virtualizing desktops and hosting them on remote servers simplifies the IT manager's tasks, such as applying security patches, performing data backups. Second, it also enables better resource management and reduces costs, since multiple desktop VMs can be hosted on a high-end server, which may still be more cost-effective than running each desktop on a separate PC. At the same time, in addition to their use for business purposes in enterprise settings, desktop VMs that are hosted in the cloud are beginning to be offered for consumer use. Notably, major cloud providers such as Amazon [2] and Microsoft [3] are currently offering Windows virtual desktops that can be accessed from a wide ranges of end devices including tablets.

The confluence of these trends—the emergence of both distributed clouds and popularity of virtual desktops—creates both opportunities and challenges. Today a virtual desktop provider needs to manually choose the best data center location for each end-user's virtual desktop. In the simplest case, each VD can be hosted at a cloud data center location that is nearest to its user (owner). However, such manual placement becomes increasingly challenging for several reasons. To start with, while this may be straightforward in cloud platforms that offer a choice of a few locations (e.g., with Amazon, one would host all VDs for US east coast users at the east coast data center), it becomes progressively more challenging as the number of locations continues to grow in highly distributed clouds that already offer a large number of locations. Additionally, different data center locations may have varying hosting capacities. Regional locations might have comparatively smaller capacities than the "global" locations; this implies that naïvely placing all VDs from a location at their nearest regional site might not be practical due to resource constraints. More interestingly, not all VDs are sensitive to network latency. Therefore, users may not see significant performance improvement when their VDs are placed at the closest location. Specifically VDs that run latency-sensitive applications such as multi-player games or video playbacks will see disproportionately greater benefit from nearby placement compared to those that run simple desktop applications such as e-mail or a text editor. Further, VDs will see dynamic workloads—users may choose to run different applications at different times and this workload mix may change over time. In addition, users may themselves move locations, particularly those that access their VDs via mobile devices, or go from work to home. This set of challenges imply that a static and manual placement of VDs at the nearest cloud location may not always be enough or even feasible. We argue that the cloud platform should incorporate intelligence to automatically determine the best location for hosting each application, and transparently and seamlessly adjust such mappings over time with changing application needs.

Towards this end, we present VMShadow, a system to transparently and dynamically manage the location and performance of virtual desktops in distributed clouds. Our system automates the process of placing, monitoring and migrating cloud-based virtual desktops across the available cloud sites based on the location of users and latency-sensitivity of the applications. VMShadow performs black-box virtual desktop fingerprinting to assign different latency-sensitive scores based on the packet-level statistics collected from hypervisor. It then employs either an ILP algorithm or a cost-aware greedy algorithm, depending on the problem scale, to pick new locations for latency-sensitive VMs that balance the cost-benefit tradeoffs. Both algorithms are able to make placement decisions while considering the existing virtual desktop locations. VMShadow executes the new VM placement plan using live migration across the WAN, optimized by techniques such as delta encoding and content-based redundancy elimination [4]. More specifically, to migrate a VM to a new location across the WAN, VMShadow first live migrates the disk and memory state of a VM using the optimized WAN live migration. In the scenario where the public IP address of the virtual desktop changes, VMShadow seeks to maintain existing TCP connections between the clients and server VMs using connection proxies. The connection proxies communicate the changes of IP address and port number and rewrite the network packet headers to ensure that the migration is transparent to applications. As a result, VMShadow allows a client to stay connected irrespective of whether the server or even the client moves, whether the client or server is behind a NAT, and whether network entities such as routers and NAT devices are cooperating.

Although VMShadow is designed to be a general platform, in this paper we employ it primarily to optimize the performance of desktop clouds, as illustrated in Fig. 1. Desktop clouds offer an interesting use-case for VMShadow since desktops run a diverse set of applications, not all of which are latency-sensitive. We implement a prototype of VMShadow in a nested hypervisor,

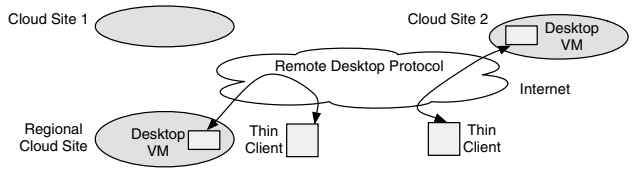


Fig. 1 Illustration of a distributed desktop cloud. Users can access their desktop VMs hosted in the regional cloud sites or global cloud sites through remote desktop protocol such as virtual network computing (VNC)

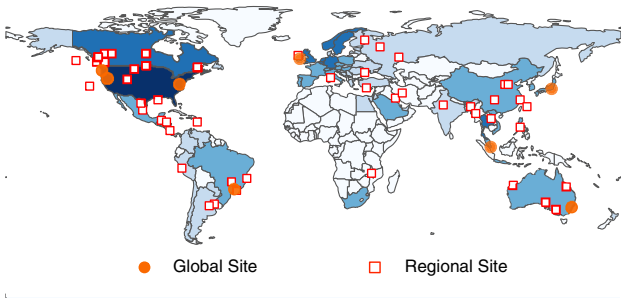


Fig. 2 A hypothetical distributed cloud. *Circles* denote global cloud location while *squares* denote regional sites

i.e., Xen-Blanket [5], and experimentally evaluate its efficacy on a mix of latency-sensitive multimedia and latency-insensitive VDs running on a Xen-based private cloud and Amazon's EC2 public cloud. Our results show that VMShadow's black-box fingerprinting algorithm is able to discriminate between latency-sensitive and insensitive virtual desktops and judiciously moves only those VDs that see the most benefit from migration, such as the ones with video activity. For example, VDs with video playback activity see up to 90% improvement in refresh rates due to VMShadow's automatic location optimizations. We demonstrate the live migration of VDs across Amazon EC2 data centers with trans-coastal VM migrations of Ubuntu desktops with 1.4 GB disk and 1 GB RAM take 4 min. We show that our connection migration proxy—based on dynamic rewriting of packet headers—imposes an overhead of 13 μs per packet. Our results show the benefits and feasibility of VMShadow for optimizing the performance of multimedia VDs, and more generally, of a diverse mix of virtual machine workloads.

2 Background

An infrastructure-as-a-service (IaaS) cloud allows application providers to rent servers and storage and to run any virtualized application on these resources. We assume that our IAAS cloud is highly distributed and offers a choice of many different geographic locations ("cloud sites") for hosting each application. For example, in Amazon's EC2, an application provider may choose to host their application at any of their global locations such as Virginia and Singapore. We assume that future cloud platforms will be even more distributed and offer a much larger choice of locations (e.g. one in each major city or country). A distributed cloud is likely to comprise heterogeneous data centers—some locations or sites will be very large ("global") data centers, while many other regional sites will comprise smaller data centers as depicted in Fig. 2. Such a heterogeneous distributed cloud maps well to how public clouds are likely to evolve—comprising of a few large global sites that offer economies of scale, while smaller regional sites offer greater choice in placing latency-sensitive applications. The model also maps well to distributed private clouds run by enterprises for their own internal needs—typical enterprise IT infrastructure consists of a few large backend data centers (to extract economies of scale by consolidating IT applications) and several smaller data centers at branch office locations (which host latency-sensitive applications locally).

We focus our attention on a single application class, namely cloud-based desktops (also referred to as desktop clouds that host a large number of VDs in data centers) that run on virtual machines (VMs) in the cloud data center. Each desktop VM represents a "desktop computer" for a particular user. Users connect to their desktop from a thin client using remote desktop protocols such as VNC or Windows RDP. We treat the VMs as black boxes and assume that we do not have direct visibility into the applications running on the desktops; however, since all network traffic to and from the VM must traverse the hypervisor or its driver domain, we assume that it is possible to analyze this network traffic and make inferences about ongoing activities on each desktop VM. Note that this black-box assumption is necessary for public clouds where the VDs belong to third party users.

To provide the best possible performance to each desktop VM, the cloud platform should ideally host each VM at a site that is nearest to its user. Thus a naïve placement strategy is to determine the physical location of each user (e.g. New York, USA) and place that user's VM at the geographically nearest cloud site. However, since nearby regional cloud cites may have a limited server capacity, it may not always be possible to accommodate all VDs at the regional site and some subset of these desktops may need to be moved or placed at alternate regional sites or at a backend global site. Judiciously determining which VDs see the greatest benefit from nearby placement is important when making these decisions.

Fortunately, not all desktop VMs are equal in terms of being latency-sensitive. As we show in Sect. 4, the

performance of certain desktop applications is significantly impacted by the geographic distance between VD and its user. While for other applications, the location is not a major factor for good performance. In particular, network games require high interactivity or low latencies; video playback or graphics-rich applications require high refresh rates or high bandwidth while using remote desktop protocol. Such applications see the greatest benefits from nearby placement since this yields low round-trip time between the user and her VM or ensures higher bandwidth or less congested links. Thus identifying the subset of desktops that will benefit from closer placement to users is important for good end-user experience. Further since users can run arbitrary applications on their desktops, we assume that VM behavior can change over time (in terms of its application mix) and so can the locations of users (for instance, if a user moves to a different office location). The cloud platform should also be able to adjust to these dynamics.

3 VMShadow design goals

Our goal is to design VMShadow, a system that optimizes the performance of cloud-based VDs via judicious placement across different sites in a distributed cloud. VMShadow seeks to dynamically map latency-agnostic VMs to larger backend sites for economies of scale and latency-sensitive ones to local (or nearby regional) sites for a better user experience. To do so, our system must fingerprint individual VMs' traffic to infer their degree of latency-sensitivity while respect the black-box assumption. Our system must then periodically determine which group of VMs need to be moved to new sites based on recent changes in their behaviors and then transparently migrate the disk and memory state of these desktops to new locations without any interruption. Typically VDs running latency-sensitive applications, such as games or multimedia applications (video playback), are the best candidates for such migration. Finally, our system should transparently address networking issues such as IP address changes when a VM is moved to a different data center location, even if the client or desktop is behind a network address translation (NAT) device.

3.1 VMShadow architecture

Figure 3 depicts the high-level architecture of VMShadow. Our system achieves the above goals by implementing four components: (1) a black-box VM fingerprinting technique that infers the latency-sensitivity of VMs by analyzing packet-level network traffic, (2) an ILP and an efficient greedy algorithms that judiciously move highly latency-sensitive VMs to their ideal locations by considering

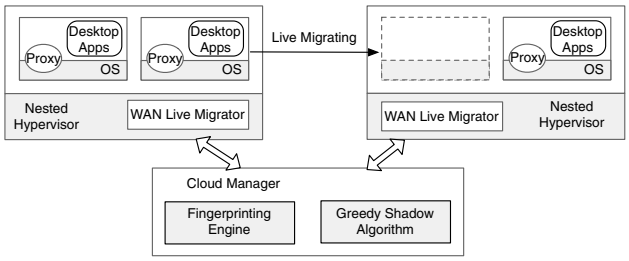


Fig. 3 VMShadow architecture. The central cloud manager performs latency-sensitivity fingerprinting for each desktop VM and employs a greedy algorithm that migrates highly latency-sensitive VMs to closer cloud sites at least cost. For each hypervisor, we implement a live migration technique that achieves WAN-specific optimizations. For each desktop VM, we use proxy to transparently migrate TCP connection

latency, migration cost as well as latency reduction. (3) An efficient WAN-based live migration of a VM's disk and memory state using WAN-specific optimizations, and (4) a connection migration proxy that ensures seamless connectivity of currently active TCP connections—despite IP address changes—in WAN live migration. We describe the design of each of these components in Sects. 4–6 and the implementation of VMShadow in Sect. 7.

4 Black-box VM latency fingerprinting

VMShadow uses a black-box fingerprinting algorithm to determine each virtual desktop's latency-sensitivity score. This approach is based on the premise that certain applications perform well, or see significant performance improvements, when located close to their users. We first describe our observations of distinct network characteristics of latency-sensitive and insensitive applications running inside virtual desktop.

4.1 Latency-sensitive applications

Consider desktop users that play games; clearly the nearer the VD is to the user, the smaller the network round-trip-time (RTT) between the desktop and the user's thin client. This leads to better user-perceived performance for such latency-sensitive gaming. Similarly, consider users that watch video on their virtual desktops—either for entertainment purposes from sites such as YouTube or Netflix, or for online education via Massive Online Open Courses (MOOCs) or corporate training. Although video playback is not latency-sensitive per se, it has a high refresh rate (when playing 24 frames/s video, for example) and also causes the remote desktop protocol to consume significant bandwidth. As the RTT between the thin client display and the remote VD increases, the

Table 1 Statistics of VNC frame response time for latency-insensitive applications

	LAN (second)				US-WEST (second)			
	Avg	Max	Min	Std	Avg	Max	Min	Std
Text editor	0.191	0.447	0.094	0.0705	0.288	0.605	0.149	0.121
Web browser	0.059	0.269	0.009	0.050	0.174	0.472	0.099	0.071

For both online text editing and web browsing, users see acceptable latencies [6]

performance of video playback suffers (see Fig. 4). Many VNC players, for instance, perform pull-based screen refresh and each refresh request is sent only after the previous one completes. Hence, the RTT will determine the upper bound on the request rate. Thus if the RTT is 100 ms (not unusual for trans-continental distances in the US), such a player is limited to no more than 10 refresh requests per second, which causes problems when video playback requires 20 or 30 frames/s. In this case, locating the VD closer to the end-user yields a lower RTT and potentially higher refresh rates and better performance. This is depicted in Fig. 4 which shows a CDF of the VNC refresh rate of a client in Massachusetts when the desktop VM is on a LAN, or at US-East and US-West sites of Amazon EC2. More specifically, in Fig. 4a, when watching YouTube, we observe about 82% of the frame requests of LAN local streaming are served in less than 41.7 ms—the update frequency for 24 FPS video. However, in Fig. 4b, when a user is watching video on the virtual desktop hosted at US-West about 70% VNC frames are updated after more than 125 ms, with the potential loss of video frames. Thus, proper placement of desktops with video applications significantly impacts user-perceived performance; similar observations hold for other application classes such as network games or graphics-rich applications.

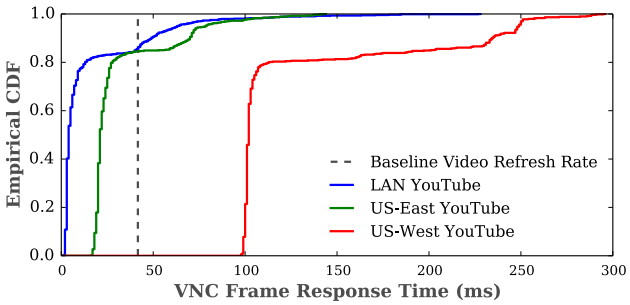
4.2 Latency-insensitive applications

In contrast to the above, applications such as simple web browsing and word processing as shown in Table 1 are insensitive to latency. Although these are interactive applications, user-perceived performance is not impacted by larger RTT since they are within the human tolerance for interactivity (as may be seen by growing popularity of cloud-based office applications such as Google docs and Office 360).

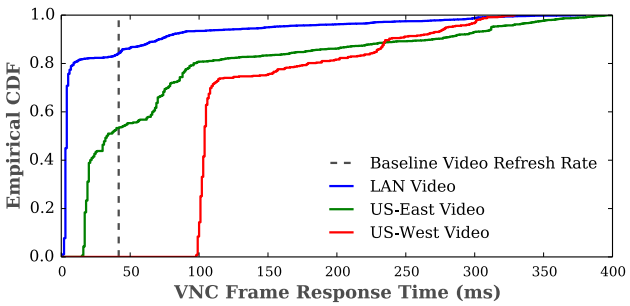
Based on our observations of different latency requirements of VD applications, we conclude that different VDs will have different degrees of latency-sensitivity depending on the collection of applications they run. Next, we will describe VMShadow's black-box latency fingerprinting algorithm that recognizes this diversity.

4.3 Black-box fingerprinting algorithm

The goal of our black-box fingerprinting algorithm is to assign latency sensitive scores S to virtual desktops based on their network characteristics without explicitly looking inside each VM. For a particular virtual desktop, the end user (via a thin client) can run arbitrary applications simultaneously. This indicates virtual desktops will show dynamic latency requirements and these requirements will be reflected in the network traffic. For a total of N virtual desktops, we are mainly interested in finding the relative latency scores for each one. We use the normalized network traffic throughput h^*, the normalized remote desktop protocol throughput e^*, and the latency-sensitive percentage of normal internet traffic f^* to infer the latency score. The rationale behind our choice of these three indicators is as follows. First, a "chatty" virtual desktop is more likely to be sensitive to the placement.



(a) Watching online video.



(b) Watching local video.

Fig. 4 CDF comparisons of VNC frame response times for latency-sensitive applications. Users have a better experience with watching videos when the VNC server is closer

Second, a virtual desktop that interacts with thin client frequently is more likely to benefit from closer placement. Third, based on our observations, a virtual desktop that runs graphic-rich applications, e.g., videos, is more likely to benefit from placement optimization.

To calculate these values for ith desktop VM, we collect packet-level traffic traces for a time window of size T_i. The traces are collected by observing the incoming and outgoing traffic of a VM from the driver domain of the hypervisor (e.g., Xen's dom0). We denote the total network traffic observed for ith VM as H_i and obtain the throughput h_i and normalized throughput h_i^* in Eqs. 1 and 2.

$$h_i = \frac{H_i}{T_i}, \tag{1}$$

$$h_i^* = \frac{h_i}{\hat{h}}, \tag{2}$$

$$\hat{h} = \max\langle h_1, h_2, \ldots h_N\rangle$$

Next, we identify the total amount of remote desktop traffic E_i using the default ports, e.g., port 5901 (server port for the VNC protocol) or port 3389 (server port for the Windows RDP protocol). Similarly, we can calculate the protocol throughput e_i and normalized throughput e_i^* in Eqs. 3 and 4.

$$e_i = \frac{E_i}{T_i}, \tag{3}$$

$$e_i^* = \frac{e_i}{\hat{e}}, \tag{4}$$

$$\hat{e} = \max\langle e_1, e_2, \ldots e_N\rangle$$

Lastly, to calculate the latency-sensitive percentage of internet traffic for ith virtual desktop f_i^*, we first use our list of latency-sensitive server ports and addresses to identify the amount of latency-sensitive traffic F_i and then obtain f_i^* as in Eq. 5.

$$f_i^* = \frac{F_i}{(H_i - E_i)} \tag{5}$$

To obtain the list of latency-sensitive server ports and addresses, we assume that the administrator provides this initial information based on prior experience. Notably, "http://youtube.com" or other online video streaming sites would be included in the initial list. VMShadow then evolves this list by adding or removing information from the list using classification results. Currently,VMShadow uses K-nearest-neighbors (KNN) classifier to label each new TCP connection as latency-sensitive or not. When building up KNN model, we represent each TCP connection as d-dimension feature vector[1] $\mathbf{g} \in \mathbf{R}^d$ and classify the new connections as latency sensitive or not based on majority vote of its K nearest neighbors. Here, we choose K to be 3. To collect training data, we manually run various selected applications (that we know of their latency-sensitivity) inside virtual desktops and collect the feature vector for each connection. If a new connection is labeled as latency-sensitive, VMShadow will then add the corresponding server port and address to the maintained list. Otherwise, the information will be removed from the maintained list if exists. Finally, we calculate the desktop VM's latency scores S in Eq. 6.

$$S = w_h * h^* + w_e * e^* + w_f * f^*, \tag{6}$$

where $W = \langle w_h, w_e, w_f\rangle$ represents the weights we assign to each normalized term. Currently, we use $W = \langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\rangle$.

Thus, VMShadow keeps track of each virtual desktop's latency score for a time-window of length M, denoted as $\langle S(t-M), S(t-M+1), \cdots S(t)\rangle$ and uses the moving average $\frac{1}{M}\sum_{j=0}^{M} S_i(t-j)$ to represent the rank of ith VD. VDs with consistently high rank become candidates for latency optimization—in cases where they are not already in the best possible data center location—as described next.

5 VMShadow algorithm

In this section, we explain VMShadow's algorithm that enables virtual desktop deployments to "shadow", i.e., follow their users through intelligent placement. Given a distributed cloud with K locations, placements of N active desktop VMs and their latency-sensitive ranks $\langle S_1, S_2 \ldots S_N\rangle$, our shadowing algorithm employs the following steps periodically.

Step 1. Identify potential candidates to move. VMShadow determines which VMs are good candidates for migration to a different location—either relocated to a closer cloud location or evicted from a regional site with limited resource. We define a high threshold S_{up} and a corresponding low threshold S_{lo} to identify VMs for either relocation or eviction. Note that we can obtain S_{up} and S_{lo} by setting up two benchmark virtual desktops, one that runs latency-sensitive applications and the other runs that latency-insensitive applications, and measure their network traffic. In particular, for ith VM with a latency score of S_i, if $S_i > S_{\text{up}}$, it becomes a candidate for relocation; if $S_i < S_{\text{lo}}$, it is a candidate for eviction. As an example, a desktop VM with consistent video or gaming activities will become

[1] Example features include throughput, connection duration or inter-packet latency.

a candidate for optimization and those that have not seen such activities for long periods will become candidates for eviction.

Step 2. Determine new locations for each candidate. For each VM that is flagged as a candidate for relocation, VMShadow next identifies potential new cloud locations for that VM. To do so, it first determines the location of the user for that desktop VM (by performing IP geo-location of the VNC thin client's IP address [7]). It then identifies the k closest cloud sites by geographic distance and then computes the network distance (latency) of the user to each of these k sites. These sites are then rank-ordered by their network distance as potential locations to move the VM. Candidate VMs that are already resident at the "best" cloud site are removed from further consideration.

Step 3. Analyze VMs' cost-benefit for placement decision. For each candidate VM for relocation, VMShadow performs a cost-benefit analysis of the possible move. The cost of a move to a new location is the overhead of copying the memory and disk state of the VM from one location to another over the WAN. The benefit of such a move is the potential improvement in user-perceived performance (e.g. latency reduction). In general, the benefit of a move is magnified if the VM has a relatively small disk and memory footprint(cost) and a high latency-sensitive rank. Since regional/local cloud sites may have smaller capacities, VMShadow must perform the cost-benefit analysis to identify VMs that yield the most benefit at the least cost. Also VMShadow could evict low-ranked VMs to free up resources when necessary. We formulate the above problem as an integer linear program (ILP) optimization in Sect. 5.1. Since an ILP can have exponential running costs, we also devise an efficient greedy heuristic that incorporates cost-benefit trade-off in Sect. 5.2.

Step 4. Trigger VMShadow migrations. The final step involves triggering migrations of the disk and memory state of VMs to their newly chosen locations. Our approach is built upon prior work CloudNet [4] that provides an end-to-end and optimized solution for live migrating virtual machines in the context of Wide Area Network. Our work extends CloudNet in two ways. First, we re-implement all optimizations inside a nested hypervisor, i.e. Xen-Blanket [5]. This is an important extension because it provides us the flexibility to live migrate virtual machines between two nested hypervisors, eliminating the needs for hypervisor privilege and cloud provider lock-in. In another words, VMShadow can seamlessly migrate virtual machines between different cloud platforms with geographically diverse data center locations. Second, we propose an alternative method to ensure TCP connections staying active after VM migrations. Unlike CloudNet [4], our method does not require specialized hardware support. Our VM and connection migration techniques are detailed in Sect. 6.

5.1 VMShadow ILP placement algorithm

In this section, we describe our ILP algorithm that places above-threshold VMs—virtual desktops that have latency scores larger than S_{up}—to a better cloud location by considering the migration cost and latency reduction. Assume we have access to K data center locations, and a total of J server hosts. Our goal is to pick the ideal data center for all I VDs within the resource constraints of the hosts. Essentially, we can translate the problem into selecting hosts with different network latencies to run the VDs.

Let $\langle U_j, M_j, D_j, N_j \rangle$ denote the available resource vector of Host_j representing CPU cores, memory, disk and network bandwidth, respectively. In accounting the available resource vector for each host, we also count the account of resource used by below-threshold VMs (scores lower than S_{lo}) that are marked for eviction. This enables us to prioritize the need of high latency-sensitive VMs in resource-constrained regional sites by moving insensitive VMs to a larger/global site. Similarly, let $\langle c_i, m_i, d_i, n_i \rangle$ denotes the resource vector of VM_i.

Let A_{ij} be the binary indicator such that:

$$A_{ij} = \begin{cases} 1 & \text{if } i\text{th VM is on } j\text{ th host} \\ 0 & \text{otherwise.} \end{cases}$$

Our goal is then to find an appropriate assignment to each A_{ij} that minimizes the sum of normalized latency, migration cost and maximizes latency reduction while satisfying the constraints. Intuitively, the new VD placement should incur low migration cost and have large latency reduction. Similarly, we use $\bar{A}$ to represent the current placement of VMs among J hosts. More specifically, let us denote the current placement of ith VM as p_i, i.e., it is running in Host_{p_i}, we then have $\bar{A}_{ip_i} = 1$ (and all other element in vector $\bar{A}_i$ as 0) for ith VM. We formulate the ILP problem as following:

$$\min \quad \sum_{i,j} \frac{A_{ij} L_{ij}}{L_i} + \sum_{i,j} \cdot \{A_{ij} = \bar{A}_{ip_i}, j \neq p_i\} \left(\frac{C_i}{C} - \frac{B_{ij}}{B} \right), \tag{7}$$

subject to:

$$\sum_{i=1}^{I} A_{ij} u_i \leq U_j, \quad \forall j = 1 \ldots J, \tag{8}$$

$$\sum_{i=1}^{I} A_{ij} m_i \leq M_j, \quad \forall j = 1 \ldots J, \tag{9}$$

$$\sum_{i=1}^{I} A_{ij} d_i \leq D_j, \quad \forall j = 1 \ldots J, \tag{10}$$

$$\sum_{i=1}^{I} A_{ij} n_i \leq N_j, \quad \forall j = 1 \ldots J, \tag{11}$$

$$\sum_{j=1}^{L} A_{ij} = 1, \quad \forall i = 1 \ldots I, \tag{12}$$

$$A_{ij} \in \{0, 1\}, \quad \forall i = 1 \ldots I, j = 1 \ldots J, \tag{13}$$

where L_i is the maximum latency of placing ith VM among all J hosts, i.e. $L_i = \max\langle L_{i1}, \ldots L_{ij}\rangle$. Specially, L_{ij} denotes the expected network latency between a thin client that connects to ith VDs and jth host. C_i and B_{ij} denote the cost and benefit of migrating ith VM from its current host to a new one. We consider the cost of migrating ith VM to be the amount of data to be moved and the benefit of migrating to host j be the latency reduction. Further, we use C and B to denote the maximum cost and benefit of migrating all I VM. That is, $C = \max\langle C_1, C_2 \ldots C_I\rangle$ and $B = \max\{B_{ij} | \forall i = 1 \ldots I, \forall j = 1 \ldots J\}$. Our objective function 7 not only considers the normalized latency associated with new placement decision, but also uses indicator function $\mathbb{1}$ to capture the relation between new placement decision and current virtual desktops to hosts mapping. We normalize each term to balance the impacts of metric on determining the placement decision. Constraints (8–11) ensure the placement decision of VMs satisfy the physical resource constraints of the hosts while constraints (12–13) together ensure each VM will only be placed in one host at every time point. While optimally, our ILP requires long time to compute placement decisions for large problem sizes. In the next section, we propose three different greedy heuristics that efficiently compute with new placement decisions.

5.2 VMShadow greedy heuristics

5.2.1 Rank-ordered greedy

In this approach, we consider all desktop VMs whose latency-sensitive rank exceeds a certain threshold S_h and consider them for relocation in rank order. Thus the highest ranked desktop VM is considered first for optimization. If the closest regional cloud site to this VM has insufficient resources, the greedy heuristic attempts to free up resources by evicting VMs that have been flagged for reclamation. If no VMs can be reclaimed or freed-up resources are insufficient to house the candidate VM, the greedy approach then considers the next closest cloud site as a possible home for the VM. This process continues until a new location is chosen (or it decides that the present location is still the best choice). The greedy heuristic then considers the next highest ranked desktop VM and so on. While rank-ordered greedy always moves the most needy (latency-sensitive) VM first, it is agnostic about the benefits of these potential moves—it will move a highly ranked VM from one data center location to another even if the VM is relatively well-placed and the move yields a small, insignificant performance improvement.

5.2.2 Cost-oblivious greedy

An alternate greedy approach is to consider candidates in the order of relative benefit rather than rank. This approach considers all VMs that are ranked above a threshold S_{up} and orders them by the relative benefit B of a move. We define the benefit metric as the weighted sum of the absolute decrease in latency and the percentage decrease. If l_1 and l_2 denote the latency from the current and the new (closest) data center to the end-user, respectively, then benefit B is computed as:

$$B = w_1 \cdot (l_1 - l_2) + w_2 \cdot \frac{(l_1 - l_2) \times 100}{l_1} \tag{14}$$

where w_1 and w_2 are weights, $l_1 - l_2$ denotes the absolute latency decrease seen by the VM due to a move and the second term is the percentage latency decrease. We do not consider the percentage decrease alone, since that may result in moving VMs with very low existing latency. For example, one VM may see a decrease from 100 to 60 ms, yielding a 40% reduction, while another may see a decrease from 2 to 1 ms, yielding a 50% reduction. Although the latter VM sees a greater percentage reduction, its actual performance improvement as perceived by the user will be small. Consequently, the benefit metric considers both the percentage reduction and the absolute decrease. The weights w_1 and w_2 control the contribution of each part—we currently use $w_1 = 0.6$ and $w_2 = 0.4$ to favor the absolute latency decrease since it has more direct impact on improving performance.

Once candidate VMs are ordered by their benefit, the cost-oblivious greedy heuristic considers the VM with the highest benefit first and considers moving it using a process similar to rank-ordered greedy approach. The one difference is that if the VM cannot be relocated to the best location, this approach recomputes the benefit metric to the next best site and re-inserts the VM into the list of VMs in benefit order, and picks the VM with most benefit. Ties are broken by rank (if two candidates have the same benefit metric, the greedy considers the higher ranked VM first).

5.2.3 Cost-aware greedy

Cost-oblivious greedy only considers the benefit of potential moves but ignores the cost of such migrations. Since the disk and memory state of VMs will need to be migrated over a WAN, and this may involve copying large amounts (maybe gigabytes) of data, the costs can be substantial. Consequently, the final variant of greedy,

known as cost-aware greedy heuristic, also considers the cost of moving a VM as:

$$C = (S_{\text{disk}} + S_{\text{mem}}) \cdot \frac{1}{1-r}, \tag{15}$$

where S_{disk} and S_{mem} denote the size of the disk and memory state of the virtual machine and parameter r captures the dirtying rate of the VM relative to the network bandwidth.[2] The dirty rate r could be either estimated by the network traffic to VD or monitored from hypervisor as the disk I/O write rates.

The cost-aware greedy approach then orders all candidate VMs using $\frac{B}{C}$ (i.e. the benefit weighted by the cost). A candidate with a higher $\frac{B}{C}$ offers a higher performance improvement benefit at a potentially lower migration cost. The VM with the highest $\frac{B}{C}$ is considered first for possible movement to the closest cloud site. Like before, if this site has insufficient server resources, then VMs marked for reclamation are considered for eviction from this site to make room for the incoming VM. Note, Eq. 15 implicitly consider the potential cost of reclamation as one has to at least free up C amount of disk and memory spaces by evicting VMs. If no such reclamation candidates are available, the VM is considered for movement to the next closest site. The benefit metric to this next site is recomputed and so is the $\frac{B}{C}$ metric and the VM is reinserted in the list of candidate VM as per its new $\frac{B}{C}$ metric. The greedy heuristic then moves on to the next VM in this ordered list and repeats. Ties are broken using the VMs' rank.

Our VMShadow prototype employs this cost-aware greedy heuristic. It is straightforward to make the cost-aware greedy implementation to behave like the cost-oblivious or the rank-ordered greedy variants by setting the cost (for cost-oblivious) and benefit (for rank-ordered greedy) computation procedures to return unit values.

Avoiding oscillations: to avoid frequent moves or oscillatory behavior, we add "hysteresis" to the greedy algorithm—once a candidate VM has been moved to a new location, it is not considered for further optimization for a certain hysteresis duration T. Similarly, any VM which drops in its latency-sensitivity rank is not evicted from a local site unless it exhibits consistently low rank for a hysteresis duration T'. Moreover, the cost-benefit metrics avoid moving VMs that see small performance improvements or those that have a very high data copying cost during migration.

[2] Live migration of a VM takes place in rounds, where the whole disk and memory state is migrated in the first round. Since the VM is executing in this period, it dirties a fraction of the disk and memory, and in the next round, we must move $(S_{\text{disk}} + S_{\text{mem}}) \cdot r$, where r is the dirtied fraction. The next round will need an additional $(S_{\text{disk}} + S_{\text{mem}}) \cdot r^2$. Thus we obtain an expression: $(S_{\text{disk}} + S_{\text{mem}}) \cdot (1 + r + r^2 + \cdots)$. This expression can be further refined using different disk and memory dirtying rates for the VM.

6 Transparent VM and connection migration

While VMShadow attempts to optimize the performance of latency-sensitive VMs by moving them closer to their users, it is critical that such moves be transparent to their users. The desktop VM should not incur downtime when being moved from one cloud site to another or encounter disruptions due to a change of the VM's network address. VMShadow uses two key mechanisms to achieve this transparency: live migration of desktop virtual machines over the WAN, and transparent migration of existing network connection to the VM's new network (IP) address. We describe both mechanisms in this section.

6.1 Live migration over WAN

When VMShadow decides to move a VD from one cloud site to another, it triggers live migration of the VM over the WAN. While most virtualization platforms support live VM migration within a data center's LAN [8], there is limited support, if any, for a migration over the wide area. Hence, we build on the WAN-based VM migration approach that we proposed previously [4], but with suitable modifications for VMShadow's needs.

The WAN-based VM migration that we use in VMShadow requires changes to the hypervisor to support efficient WAN migration. It is possible to implement these modifications of the hypervisor in private clouds where an enterprise has control over the hypervisor. Similar modifications are also possible in public clouds where the cloud provider itself offers a desktop cloud service to users. However, the desktop cloud service may also be implemented by a third party that leases servers and storage from a public IaaS cloud provider, e.g., derivative clouds [9, 10]. In such scenarios, the third party should not expect modifications to the hypervisor.

To support such scenarios also, we employ a nested hypervisor to implement VMShadow's migration techniques. A nested hypervisor runs a hypervisor h' inside a normal virtual machine that itself runs on a typical hypervisor h; actual user VMs run on top of hypervisor h'. Since the nested hypervisor is fully controlled by the desktop cloud provider (without requiring control of the underlying hypervisor), it enables hypervisor-level optimizations. Note that using a nested hypervisor trades flexibility for performance due to the additional overhead of running a second hypervisor; however, Xen-Blanket [5], which we use in our prototype has shown that this overhead is minimal. As a

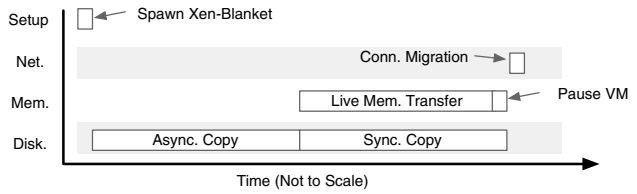


Fig. 5 VMShadow migration phase using Xen-Blanket. Upon WAN live migration, a Xen-Blanket (nested hypervisor) VM is spawned first to receive disk and memory state from source WAN live migrator. It is then followed by a live memory and disk transfer before briefly pausing the VM. The VM is successfully migrated to the new Xen-Blanket and ready to use after executing connection migration protocol

result, VMShadow can run over unmodified public cloud instances, such as Amazon EC2, and live migrate desktop VMs from one data center to another. In addition, VMShadow's WAN migration needs to transfer both the disk and memory state of the desktop virtual machine (unlike LAN-based live migration which only moves the memory state since disks are assumed to be shared). VMShadow uses a four step migration algorithm, summarized in Fig. 5.

Step 1: VMShadow uses Linux's DRBD module to create an empty disk replica at the target data center location. It then begins to asynchronously transfer the disk state of the VM from the source data center to the target data center using DRBD's asynchronous replication mode. The rate of data transfer can be controlled, if needed, using Linux' traffic control (tc) mechanisms to avoid any performance degradation for the user during this phase. The application and VM continue to execute during this period and any writes to data that has already been sent must be re-sent.

Step 2: Once the disk state has been copied to the target data center, VMShadow switches the two disk replicas to DRBD's synchronous replication mode. From this point, both disk replicas remain in lock step—any disk writes are broadcast to both and must finish at both replicas before the write returns from the disk driver. Note that disk writes will incur a performance degradation at this point since synchronous replication to a remote WAN site increases disk write latency.

Step 3: Concurrent with Step 2, VMShadow also begins transferring the memory state of the VM from the source location to the target location. Like LAN-based live migration approaches, VMShadow uses a pre-copy approach which transfers memory pages in rounds [8]. The first round sequentially transfers each memory page from the source to the destination. As with the disk, VMShadow can control the rate of data transfer to mitigate any performance impact on front-end user tasks. Since the application is running, it continues to modify pages during this phase. Hence, each subsequent round transfers the only pages that have been modified since the previous round. Once the number of pages to transfer falls below a threshold, the VM is paused for a brief period and the remaining pages are transferred, after which the VM resumes execution at the destination.

Since substantial amounts of disk and memory data need to be transferred over the WANs, VMShadow borrows two optimizations from our prior work [4] to speed up such transfers. First, block and page deltas [11] are used to transfer only the portion of the disk block or memory page that was modified since it was previously sent. Second, caches are employed at both ends to implement content-based redundancy(CBR) [4, 12]—duplicate blocks or pages that have been sent once need not be resent; instead a pointer to the cached data is sent and the data is picked up from the receiver cache. Both optimizations have been shown to reduce the amount of data sent over the WAN by 50% [4].

Step 4: Once the desktop VM moves to a new data center, it typically acquires a new IP address using DHCP. Changing the IP address of the network interface will cause all existing network connections to break and disrupt user activity. To eliminate such disruptions, VMShadow employs a connection migration protocol to "migrate" all current TCP connections transparently to the new IP address without any disruptions (TCP connections see a short pause during this transfer phase but resume normal activity once the migration completes). The connection migration is triggered after desktop VM is successfully migrated and then paused. Immediately afterwards, VMShadow updates the new mapping rules at proxies. Once the rules are updated, the migrated VM will be resumed with the new public IP address, and all subsequent packets will be rewritten. In summary, the actual traffic switching occurs after the connection migration protocol is successful. Once both the VM and connection migration phases complete, the desktop VM begins executing normally at the new cloud location. We describe VMShadow's connection migration protocol next.

6.2 Connection migration protocol

Different cloud locations are typically assigned different blocks of IP addresses for efficient routing. As a result, when a VM moves from one cloud location to another, it is typically assigned an IP address from the new location's IP block and will not retain its original IP address. This will cause TCP connections to be dropped and result in disruptions to end users' sessions. To prevent such disruptions, VMShadow employs a connection migration protocol that "migrates" these connections to the new IP address.

The issue of mobility, and having to change the IP address as a result, is a well known problem. There have been several proposals including HIP [13], LISP [14], ILNP [15] and Serval [16] that try to address this problem by separating the host identifier from the network address.

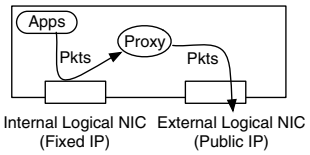


Fig. 6 Illustration of proxy IP bridging. Inside each VM, the proxy bridges an internal logical NIC with the external one, masking the potential IP address changes from the higher-level applications

With these approaches, the application connects at the TCP layer using the host identifier, while the packets are routed using the network address. When the user (i.e. host) moves, the network address changes, but the host identifier stays the same. As a result, TCP connections are not disrupted. Unfortunately, all these approaches require modifications to the application to take advantage of seamless mobility.

Instead, here we take a more pragmatic approach so that VMShadow works seamlessly with existing applications as they are. VMShadow makes use of a local proxy to implement a network connection migration protocol. VMShadow assumes that both end-points for every active connection on the migrated VM run this proxy (thus, both the thin client and the desktop VM need to run the proxy, as do other servers elsewhere with active TCP connections to the desktop VM). However, in the cases where we do not have control over servers, for example YouTube streaming servers, we can set up in-network proxy servers that are closer to VDs. We envision the virtual desktop cloud providers will be in charge of maintaining these proxy servers. In summary, as long as the proxy is in the data path for the TCP connection between end points, it can mask any address changes by dynamically re-writing the IP headers of the packets.

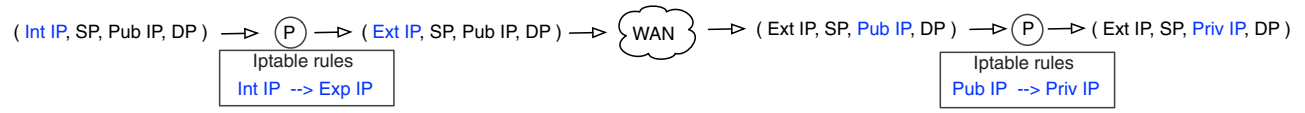
To ensure transparency, the desktop VM uses two logical network interfaces: an internal interface with a fixed, private IP address and an external interface with the "real", but potentially changing, IP address. All socket connections are bound to the internal interface as the local source address; as a result, active socket connections never directly see the changes to the external IP address. The proxy acts as a bridge between the internal and external network interfaces for all packets as shown in Fig. 6. Internally generated packets have a destination address that is the external IP address of the remote end host.

The proxy employs dynamic rewriting of packet headers (analogous to what is done in NAT devices) to bridge the two interfaces. For all outgoing packets, the default rewriting rule replaces the source IP of the internal interface with that of the external interface: $(IP_{\text{int}}, *) \rightarrow (IP_{\text{ext}}, *)$. Thus when the external IP address changes after a WAN migration, the rewrite rule causes any subsequent packet to have the new external IP address rather than the old one. Incoming packets headers are rewritten with the reverse rule, where the current external IP address is replaced with the fixed internal IP.

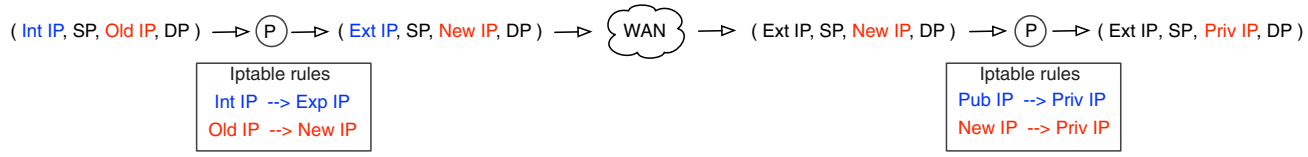
After an IP address change of a desktop VM, other end-points with connections to the desktop VM will begin seeing packets arriving from the new external IP address. However, connections on these machines expect packets from the old external IP address of the desktop VM. To ensure transparent operation, the local proxies in other end-points intercept packets with the desktop VM and apply new rewrite rules beside the default one. For example, with new rewrite rules, incoming packets arriving from the desktop VM are rewritten as $(IP_{\text{new}}, *) \rightarrow (IP_{\text{old}}, *)$ while outgoing packets to the desktop VM see rewrites to the destination IP address as $(*, IP_{\text{old}}) \rightarrow (*, IP_{\text{new}})$. These two rules ensures that outgoing packets go to the new address of the desktop VM (and thus are not lost), while incoming packets from the new IP address are rewritten with the old address before delivery to applications (that are still given the illusion of communicating with the old IP address). We illustrate various scenarios in Fig. 7.

To achieve this transparent migration, the proxies at both end points use control messages to signal each other about the change in IP address. This is done by having the desktop VM send a cryptographically signed message to the corresponding proxy informing it of the IP address change. The cryptographic signing avoids malicious third-parties from sending bogus IP address change messages and causing a denial of service. A typical IP address change control message will include the old IP address and request subsequent packets to be sent to the new address.

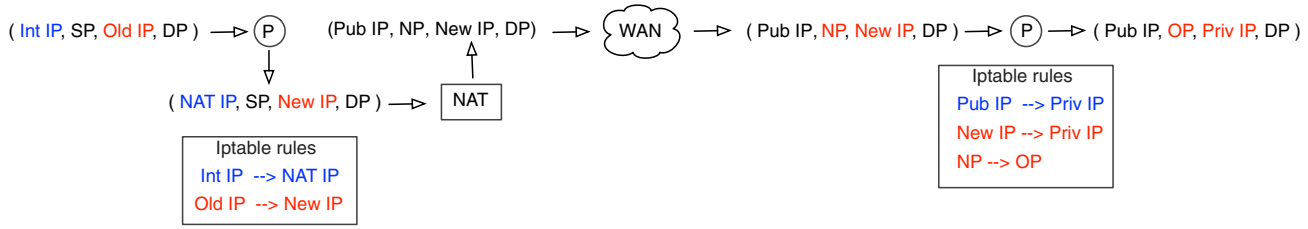
Note that the connection migration protocol is symmetric—it assumes an fixed internal interface and an external interface on all machines. Thus, the protocol can also handle IP address changes of the thin client or other machines that the desktop VM communicates with. Further, the extra rewrite rules are generated on a per-socket basis rather than a per-machine basis to support dynamic connection setup. In particular, connections established before the IP address change requires rewriting based on both default and extra rules to maintain connectivity. Connections opened after the address change talk to the new address and only need default packet rewriting. However, for incoming packets, we use the port information of the connections to distinguish between ones that need a re-write (connections opened prior to the change) versus those that do not (those opened after the change). A general rewrite rule of an outgoing packet is of the form: $(\text{IP}_{\text{int}}, \text{srcPort}, \text{IP}_{\text{old}}, \text{dstPort}) \rightarrow (\text{IP}_{\text{ext}}, \text{srcPort}, \text{IP}_{\text{new}}, \text{dstPort})$.



(a) Both entities have public IP addresses.



(b) One entity's public IP changes.



(c) A behind-NAT end point tries to communicate to an entity with new public IP address.

Fig. 7 Dynamic rule-based packet headers' rewriting sequences. We use four-tuple, i.e., the source IP, source port, destination IP and destination port, to represent a packet. Packets are matched based on the iptable rules of the proxy (default rule is in rule while new rules are in *red*). This ensures that high-level applications only see fixed internal IP without breaking the TCP connections. (Color figure online)

6.2.1 Handling NAT devices

Our discussion thus far assumes that all end points have a publicly routable IP address. However, in many scenarios, one or both end-points may be behind NAT devices. We first consider the scenario where the thin client is behind a NAT (e.g., in a home) while the desktop VM resides in a public cloud and has a public IP address. In this case, when the desktop VM is moved from one location to another, it will no longer be able to communicate with the thin client since the NAT will drop all packets from the new IP address of the desktop. In fact, the desktop VM will not even be able to notify the proxy on the thin client of its new IP address (since a "strict NAT" device drops all packets from any IP address it has not encountered thus far). To address this issue, we resort to NAT hole punching [17], a method that opens ports on the NAT to enable the desktop VM to communicate with the thin client.

VMShadow's NAT hole punching is part of the connection migration process. It works by notifying the client proxy of the IP address change from the old IP address of the desktop VM. In some scenarios, the desktop VM may be able to determine its new IP address at the destination before it migrates. This may be possible in enterprise private clouds where an IP address is pre-allocated to the VM, or even in public clouds where one can request allocation of an elastic IP address independent of VM instances. In such cases, the proxy on the desktop VM notifies the proxy on the thin client of its future IP address and requests hole punching for this new IP address. In scenarios where the IP address cannot be determined a priori, we assume that the newly migrated VM will notify the driver domain of the nested hypervisor at the old location of its new address. The driver domain can use the old IP address to notify the proxy at the thin client of the IP address change and consequently request hole punching.

Once the new IP address has been communicated to the client proxy, it proceeds to punch holes for each active socket port with the desktop VM. This is achieved by sending a specially marked packet from each active source port to each active destination port but with the new IP address as the destination IP of these specially marked packets. These packets causes the NAT device to open up these ports for accepting packets from the new IP address of the desktop VM. NAT devices typically rewrite the source port number with a specially allocated port number and create a forwarding rule; packets arriving on this NAT port are forwarded to the source port at the thin client device. Thus, a regular outgoing packet from the client to the desktop VM will see the following rewrites: the source proxy performs the first rewrite (IP_{int}, srcPort, IP_{old}, dstPort)→(IP_{NAT}, srcPort, IP_{new}, dstPort). The NAT device then further rewrites this packet as ($IP_{NAT_{Ext}}$, natPort, IP_{new}, destPort).

When the first specially marked packet of this form is received at the desktop VM, it creates a mapping of the old natPort of the source to the new natPort. Then port numbers of any outgoing packets are rewritten by replacing the old natPort with the new natPort created by the hole punching.

Note that the specially marked hole punching packet is only processed by the proxy and then dropped and never delivered to the application. In our implementation, we simply assign a TCP sequence number of 1 and have an iptables rule for dropping potential RST packet. This extension enables the connection migration protocol to work even when one of the end-points is behind a NAT device. The protocol can be similarly extended with hole punching packets in both directions when both end-points are behind NAT devices. Note in this scenario, the entity that moved from one NAT to another will need to find out the IP address of the new NAT device first before proceeding hole punching. We omit the details here due to space constraints.

7 VMShadow implementation

We have implemented a prototype of VMShadow using Linux 3.1.2 and modified Xen-Blanket 4.1.1 [5]. Our prototype is written in C and Python and consists of several interacting components as shown in Fig. 3. In the following, we describe the design trade-offs, functionalities and implementation details of each component.

7.1 Fingerprinting engine

Our fingerprinting engine includes a distributed traffic collector in each host and a central fingerprinting engine running inside the cloud manager. Its main tasks include collecting network-level traffic information from each host and calculating the latency-sensitive score for each virtual desktop. We implement the traffic collector component in Xen-Blanket's driver domain (dom0). It uses python interfaces to the Linux netfilter library, more specifically libnetfilter_queue to copy packets queued by the kernel packet filter into user-space for analysis; it periodically samples the traffic and sends the statistics to the fingerprinting engine running inside the cloud manager. The hypervisor-based fingerprinting system has negligible overhead, and does not interfere with a virtual desktops' normal performance. Specifically, the overhead can be broken down into copying packets, generating and sending statistics to the fingerprinting engine. The dominating overhead comes from copying every network packet, but can be dramatically reduced by mapping kernel buffers to user space. This allows sharing buffers between kernel and user space applications, and essentially achieving zero copying overhead. The caveat is the kernel needs to support zero-copy optimizations. For each active virtual desktop, cloud manager then analyzes the normalized network traffic, normalized protocol traffic and percentage of normal internet traffics (as described in Sect. 4) based on the collected network traffics and the maintained list of latency-sensitive ports and server addresses. A relative latency-sensitive scores is assigned to each virtual desktop at the end of fingerprinting process.

7.2 WAN live migrator

Our WAN live migrator takes any running virtual desktop and migrates them to a different host as fast as possible without disrupting its functionalities. We implement the migrator on top of the nested hypervisor, i.e., the Xen related code in Xen-Blanket, by modifying live migration code in Xen. More specifically (refer to Fig. 5 for a pictorial detail), we include DRBD-based disk state migration to concurrently transfer virtual machine disk asynchronously. For transferring memory, we employing multiple optimizations, i.e., zero page, memory page deltas and content-based redundancy elimination [4] to optimize the transferring over WAN. To mitigate the live migration impact's on the client traffic, we also implement the rate control mechanisms to control the rate of state transfer over WAN links.

7.3 Connection proxy

Our connection proxy implements our connection migration protocol discussed in Sect. 6.2 as a python process. We design and implement the proxy in a way that is easy and flexible to run in any end points such as the VDs and the thin clients. The proxy listens on a well-known port, to receive (and send) cryptographically signed messages for announcing IP address changes. It uses the `libnetfilter_queue` library to intercept outgoing and incoming packets and rewrites the corresponding TCP headers as specified by the current rewrite rules in `iptable`. Packets are reinserted into the queue once the headers have been rewritten. We use the python `scapy` library to generate the appropriate packets for NAT hole punching. Our choice of implementing the proxy in user-space is based on the trade-off of the implementation ease and overhead compared to a kernel implementation. In production use where efficiency has higher priority, one should implement the protocol in the kernel space to reduce the data copy overhead as well as `iptable` rules matching. We evaluate the overhead of our user-space proxy in Sect. 8.5.

7.4 Cloud manager

We use a centralized-architecture in implementing the cloud manager that runs periodically. It has interfaces to both fingerprinting engines and WAN live migrators that run distributed on each host. After each time period, our cloud manager feeds the latency-sensitive scores calculated by fingerprinting engine to the algorithm engine, to figure out the new virtual desktop placement. Our algorithm

engine implements both the ILP and cost-aware greedy algorithm. The migration manager then compares the new placement plan with the current placement to figure out a migration table that has three columns, i.e., the source host, the destination host and the target virtual desktop. Each row in the table represents a migration that needs to be actuated to improve the user-perceived performance. Our migration manager execute the specified migrations by contacting the WAN live migrator as well as the connection proxy on each source host. To avoid unnecessary performance degradation, our cloud manager employs two intuitive methods that both aims at reducing the percentage of live migration bandwidth usage. The first one is to limit the number of concurrent live migrations between the same hosts and the second one is to control the total bandwidth usage of live migration.

8 Experimental evaluation

In this section, we first describe our experimental setups and then present our experimental results. In designing our experiments, we are interested in answering the following key questions.

1. How accurate is our black-box fingerprinting algorithm in distinguishing latency-sensitive desktop VMs from the rest?
2. What is our proposed cost-aware greedy algorithm compared to ILP in optimizing the location of desktop VMs?
3. What is the potential overheads of using live migration to move desktop VMs from one cloud location to another and the performance benefits to desktop VMs' users?
4. How efficient is our connection migration proxy in seamlessly transferring the TCP connections?
5. Lastly, how does our prototype VMShadow work in resolving complex scenarios by detecting latency-sensitive desktop VMs and improving their performances within resource constrains?

8.1 Experimental setup

The testbed for our evaluation consists of hybrid clouds with a private cloud in Massachusetts and Amazon EC2 public clouds across different locations as shown in Fig. 8. The private cloud consists of 2.4 GHz quad-core Dell servers running Centos 6.2 and GNU/Linux kernel 2.6.32. On Amazon EC2, we use extra-large instances (m3.xlarge), each with 4 VCPUs, at two sites: US-West in Oregon and US-East in Virginia. All machines run modified Xen-Blanket 4.1.1 and Linux 3.1.2 as Dom0.

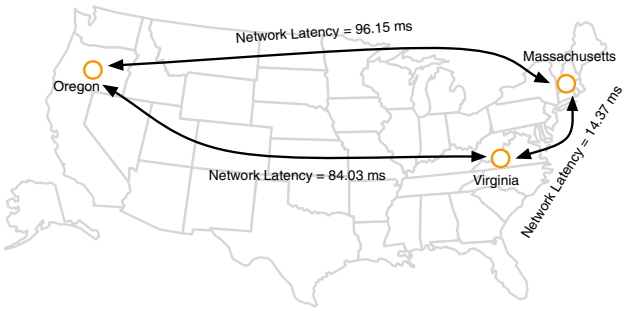


Fig. 8 Illustration of cloud sites setup in our experiments. Three cloud sides used for our experiments: a private cloud side in Massachusetts, and EC2 sites in Virginia and Oregon

Our desktop cloud consists of Ubuntu 12.04 LTS desktops that are installed with vnc4server as the VNC server. Each desktop VM will have only one desktop session and accept connections from one thin client. We use VNC as the remote desktop protocol[3] for the ease of instrumenting the according implementations. For laptop-based thin client machines, we use a modified open-source version of python VNC viewer to automate the experiment processes. For mobile phone, i.e., iPhone, thin client, we use VNC viewer acquired from App Store and manually perform user activities. Users that connect to the Ubuntu desktop will be able to run a variety of desktop applications, including OpenOffice for editing documents, Google Docs for online editing, Chrome browser[4] for web browsing and watching various online streaming, i.e., Youtube, Hulu and Netflix, Thunderbird email client and Movie Player for local video playback. Each desktop VM is assigned 1 GB memory, 1 VCPU and has a 8 GB disk of which 1.32 GB is used and runs inside Xen-Blanket dom0.

8.2 Accuracy of black-box VM fingerprinting

Black-box VM fingerprinting provides us the latency-sensitive scores for each desktop VMs without peeking inside the user actives. In this experiment, we first show that VDs with different latency-sensitive requirement exhibit vastly different network-level characteristics and then we demonstrate that our approach is able to assign correct relative latency-sensitivity scores to different VMs running various applications. We use Wireshark running on Dom0 of Xen-Blanket to collect packet-level traces for each VMs during the experiment periods.

[3] Our focus is not on comparing the performance differences of different remote desktop protocol.

[4] We choose Chrome browser due to the fact that Netflix is not supported in the default Firefox browser.

Table 2 Characterization of desktop VMs' network activities

	Uplink traffic				Downlink traffic			
	Youtube	Local video	Browsing	Text edit	Youtube	Local video	Browsing	Text edit
Non-VNC traffic (%)	37.7	0	0.67	0	53.7	0	0.62	0
Non-VNC bandwidth (KB/s)	1.85	0	0.0083	0	63.6	0	0.0059	0
Total bandwidth (KB/s)	74.6	54.5	17.94	17.14	65.8	1.54	0.454	0.86

Virtual desktops running different applications exhibit different network characteristics, i.e., remote protocol traffic and Internet traffic

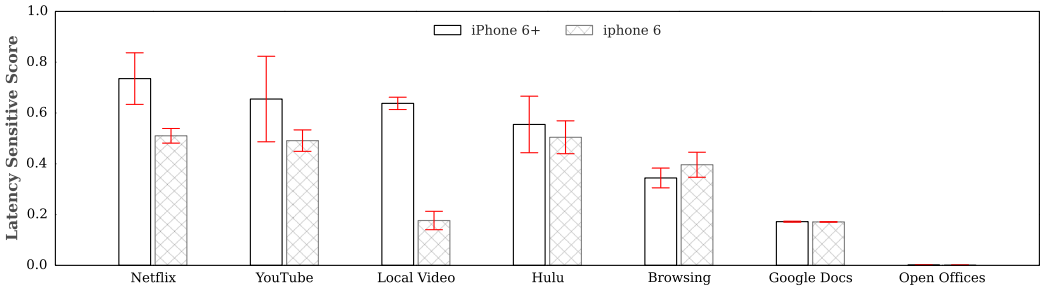


Fig. 9 Comparison of latency sensitive scores for different desktop VMs. Online streaming applications have higher scores compared to the other types of applications for both mobile clients. For both clients, all but one latency score match our hypothesis. But this "out-of-order" ranking can be remedied using threshold scores as discussed in Sect. 5

8.2.1 Characterizing network activities of desktop VMs

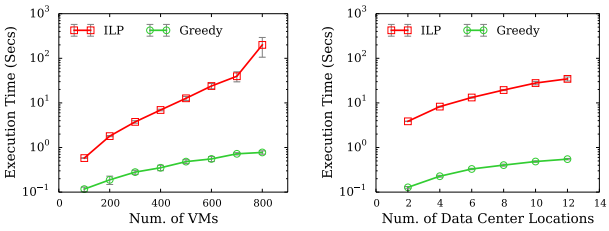
We use VNC viewer from our laptop-based thin client to perform four distinct types of user actives, i.e., watching Youtube video, browsing graphic-rich websites, text editing using OpenOffices and watching video locally on the desktop VM. In each case, we sample the traffic generated by the VM in a 3-min measurement window after a warmup period and then repeat the process five times. We compute the average statistics across all five-runs and use them to characterize the network activities of each VMs.

Table 2 summarizes the different network activities of desktop VMs for these four activities. As expected, YouTube viewing consumes higher network bandwidth both from YouTube servers and for the remote desktop protocol display; video playback from a local file does not consume network bandwidth, but the data transfer for VNC is still high due to the video playback. Web browsing and text editing consume very little bandwidth. Note in our fingerprinting algorithm, both YouTube and graphic-rich browsing will be labeled as latency-sensitive traffic based on the server ports and addresses.

8.2.2 Assigning latency-sensitive scores to desktop VMs

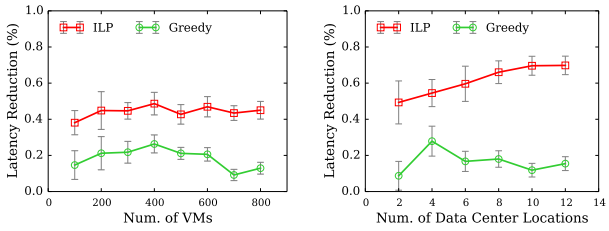
Based on the above observations, we next evaluate our fingerprinting algorithm that favors and assign relative high scores to VMs based on their latency sensitivities. We use two models of iPhones, i.e. iPhone 6 and iPhone 6+, as our mobile thin clients and collect necessary network-level data using the same setup as in previous experiment. Our measurement data, together with our list of latency sensitive ports and server addresses, are provided as input to our fingerprinting algorithm in Sect. 4 to calculate the scores.

Figure 9 compares the different latency scores assigned for desktop VMs that are running various applications. In general, our fingerprinting algorithm is able to assign "correct" relative scores to VMs running different applications. Specifically, both online streaming applications, disregarding the service providers, and local video playback are assigned with high latency scores. However, the score of watching local video using iPhone 6 is much lower than its counterparts. Recall that the relative latency score is calculated based on normalized throughput, protocol throughput and latency-sensitive throughput. Because local video does not generate internet traffic, it has a lower relative score compared to online streaming. In addition, with adaptive bitrate streaming, the amount of data transferred depends on screen size. This means iPhone 6 with smaller screen will have lower relative score than iPhone 6+. Since it heavily relies on the application bandwidths demand in calculating the scores, the results will be biased for thin clients with different screen size. To further improve the accuracy of latency scores, we could apply the algorithm based on the screen size of thin clients. Graphic-rich browsing, such as "imgur.com", is considered more latency-sensitive compared to online editing using Google docs. Lastly, local editing, with a score of 0.001, is regarded to be not



(a) The number of cloud locations is set to 40. **(b)** The number of VMs is set to 2000.

Fig. 10 Execution time comparisons between ILP and greedy algorithms. In general, greedy algorithm takes significant less time compared to ILP algorithm. For both algorithms, as the number of VMs to be assigned or the number of candidate cloud locations to be picked increase, the running time increases accordingly



(a) The number of cloud locations is set to 40. **(b)** The number of VMs is set to 2000.

Fig. 11 Comparison of latency reduction percentage between ILP and greedy algorithms. When the number of VMs to be assigned increase, the reduction is bounded by the collections of data center locations. As the number of data center locations increases, ILP is able to utilize the data center locations to find optimal solutions for each VMs

sensitive to latency at all and is potential candidate for resource reclamation.

8.2.3 Result

Desktop VMs that run different applications exhibit different level of network activities. Based on this observation, our fingerprinting algorithm is able to correctly favor and distinguish latency-sensitive desktop VMs, i.e. the ones that run online streaming or local video playback, from non-sensitive desktop VMs, i.e, local text editing.

8.3 Comparing greedy shadow algorithm to ILP

In this experiment, we study the performance differences between our cost-aware greedy algorithm and integer linear program (ILP) algorithm that is able to provide optimal results but with higher execution time. Both algorithms are implemented in VMShadow's Cloud Manager. Specifically, we implemented the ILP algorithm using Python's Convex optimization package CVXOPT that aims to minimize the latency reductions. We compare the greedy algorithm with the ILP approach in terms of scalability, i.e. execution time, and effectiveness, i.e. latency decrease percentage of desktop VMs.

To stress test both algorithms, we create synthetic scenarios with increasing numbers of desktop VMs and cloud locations and measure the execution time and effectiveness of both algorithms. In one case, we fix the number of desktop VMs to 2000 and vary the number of available cloud locations from 2 to 12. In another case, we fix the number of cloud locations to 40 and vary the number of desktop VMs in the cloud from 100 to 800. For each scenario, we run both version of algorithms ten times by assigning uniformly generated latency-sensitive scores to each VM and uniformly pick a set of data centers from a pool of forty locations. We use the average results across all runs to represent the performance and effectiveness for each scenario.

Figure 10 compares the execution time of these two algorithms in these two cases separately. As expected, the execution time of the ILP approach increases significantly with both increasing location choices (as in Fig. 10b) and increasing VMs (as in Fig. 10a); the execution time of the greedy approach, in comparison, remains flat for both scenarios. Figure 11 evaluates the effectiveness of the two algorithms in reducing the latency of desktop VMs via migrations. Our latency reduction achieved by our greedy approach is within 51–56% of the "optimal" ILP approach when our greedy algorithm has access to all forty data center locations. In the case of assigning all 2000 VMs to cloud locations, the effectiveness of our greedy approach is impacted either by the limited amount of cloud locations, (in the case of only two cloud locations), or the complexity growths. In general, the ILP approach is a better choice for smaller settings (where it remains tractable), while greedy is the only feasible choice for larger settings. Note also that our experiments stress test the algorithms by presenting a very large number of migration candidates in each run. In practice, the number of candidate VMs for migration is likely to be a small fraction of the total desktop VMs at any given time; consequently the greedy approach will better match the choices made by the ILP in these cases.

8.3.1 Results

VMShadow's greedy algorithm is able to achieve around 51–56% effectiveness with marginal execution time compared to "optimal" ILP approach, even presented with a large number of migration candidates and potential cloud locations.

Table 3 Comparison of transcontinental WAN migration of desktop VMs

	Word + YouTube	Word
Mem (GB)	0.56	0.54
Disk (GB)	1.36	1.34
Migration time (s)	165	149
Pause time (s)	2.48	2.8

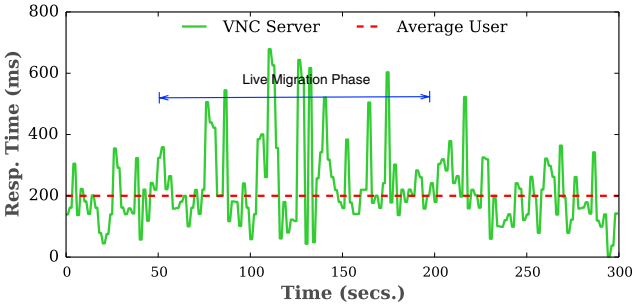
Desktop VMs are migrated from Amazon EC2's Oregon data center to Virginia data center using VMShadow that is optimized with delta-based and CBR techniques. We observe a slightly more memory and disk data transfer for desktop VM that runs more applications. The migration pause time is due to the last iteration of memory transferring and TCP connection migration

8.4 Live migration and virtual desktop performance

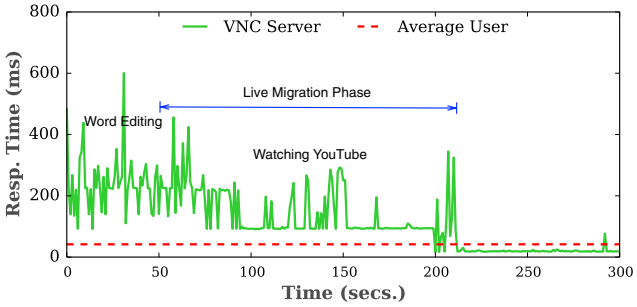
VMShadow's WAN live migrator actuates the migration decisions generated by the greedy shadow algorithm on hybrid cloud platforms by leveraging nested virtualization. In this experiments, we study the overheads of our WAN-based live migration approach, in terms of migration costs, as well as the user perceived performance benefits. We use the two Amazon sites in Oregon (US-West) and Virginia (US-East) for this experiment. The thin client is located in the Massachusetts private cloud. We run two desktop VMs in US-West. The first desktop represents a user running a text editing application for the first 50 s and then watching a YouTube video, while the second desktop represents a user only performing word editing. We perform live migration of both VMs at $t = 50$ s from Oregon data center to Virginia one, which is a site closer to the Massachusetts-based thin client, with the help of VMShadow's WAN migration component. For each live migration, we measure the total amount of data transferred and the time taken for the live migration as well as the time intervals between every VNC frame request and update.

As shown in Table 3, the delta-based and CBR optimizations used by VMShadow allow WAN migrations to be efficient; VMShadow migrates desktop VMs with 1 GB memory coast-to-coast in less than 165 s. It is useful to note that the pause time (i.e., the time when a user may perceive any unresponsiveness) for the applications as a result of the migration is relatively small, between 2.5 and 2.8 s. The total migration time is determined by how much memory to transfer and how fast memory is dirtied. Therefore, it takes longer to migrate the virtual desktop that runs YouTube than Word editing. But for the pause time, it is determined by the amount of dirty memory to transfer in the last iteration. When watching YouTube video, data is being streamed and prefetched before the last iteration. Thus, the downtime of the YouTube virtual desktop is sightly shorter.

Figure 12 shows the response time before and after the migration for both desktop VMs. We define the response time to be the time interval between sending a refresh request and receiving a response. Therefore, the lower the response time, the higher the refresh rate. Note also that the VNC player only sends a refresh request after receiving a response to its previous request. Thus the response time for such players is upper bounded by the network round-trip time. As shown in the Fig. 12a, initially the refresh rate is low since word editing does not require frequent refresh. The refresh rate increases when the user begins watching YouTube, but the refresh rate is bounded by approximately 100 ms RTT between Oregon and Massachusetts, which limits VNC to no more than 10 refreshes per second (which is not adequate for 20FPS Youtube video). Once the VM has migrated from US-West to US-East, the RTT from the thin client to the desktop VM drops significantly (and below the dotted line indicating the minimum refresh rate for good video playback), allowing VNC to refresh



(a) Latency-sensitive desktop VM.



(b) Latency-insensitive desktop VM.

Fig. 12 Comparison of WAN live migration's impact on desktop VMs running various applications. After migration, latency-sensitive desktop VM that runs online streaming achieves higher VNC frame update frequency due to lower RTT, directly improving user experience. On the other hand, latency-insensitive desktop VM that runs text editing application does not see a obvious improvement after migration

Table 4 Per-packet proxy overhead

	Total time	Copy time	Rewrite time
Average (ms)	3.375	3.36	0.0133
Std. Dev.	0.022	0.034	0.0042

We average the data copying and header rewriting

the screen at an adequate rate. Figure 12b depicts the performance of the Word editing desktop before and after the live migration. As shown, word editing involves key- and mouse-clicks and do not require frequent refreshes due to the relatively slow user activities. Thus, the refresh rate is once every few hundred milliseconds; further a 100 ms delay between a key-press and a refresh is still tolerable by users for interactive word editing. Even after the migration completes, the lower RTT does not yield a direct benefit since the slow refresh rate, which is adequate to capture screen activities, is the dominant contribution to the response time.

8.4.1 Results

Migrating a desktop VM trans-continentally takes about 4 mins depending on the workload while incurring 2.5–2.8 s pausing time. Further, not all desktop applications see benefits from migrating to a closer cloud site, demonstrating our premise that not all desktop applications are latency-sensitive.

8.5 Connection migration proxy overhead

Our connection migration proxy handles the TCP connection migration in the case of public IP address change caused by WAN live migration. In this experiment, we evaluate the overhead of running our proxy at each desktop VM, specifically the overheads of processing each packet and rewriting their headers. To conduct this micro-benchmark, we have the desktop VM connect to a server machine and establish an increasing number of TCP socket connections. The desktop VM then sends or receives 10,000 packets over each socket connection and record the overheads incurred by the proxy as we increase the number of concurrent socket connections from 8 to 64. For each measurement data, We repeat this experiment for 10 times to gather all the measurement data for results in Table 4 and Fig. 13.

The proxy overhead includes (1) data copying overhead incurred by libnetfilter Queue in copying packets from kernel space to user-space and copying back to re-insert packets, (2) matching a packet to rewrite rules, and (3) rewriting packet headers. Table 4 depicts the per-packet overhead incurred by the proxy across all runs. As shown in the table, our user-space proxy adds a 3.37 ms processing latency to

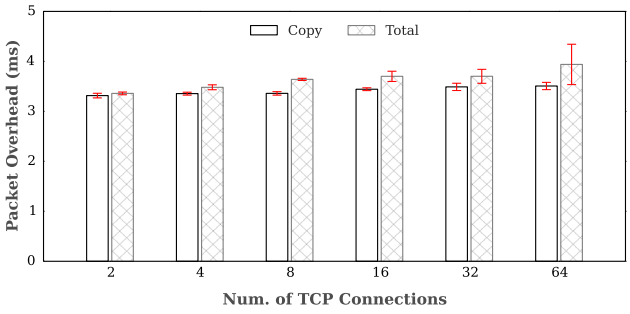


Fig. 13 Proxy processing overhead of TPC packet. Proxy overhead comprises copying network data between kernel and user-space and manipulating packets, i.e., iptable rule matching and header rewriting. The *left bar* group demonstrates the dominating copying overhead that is relatively constant to the number of active TCP connections

each outgoing and incoming packet, and 13.2 µs packet header rewriting-related latency. This means that 98.5% of the additional latency is due to the overhead of copying packets between kernel and user space; the table shows a mean 3.36 ms overhead of data copying. This overhead can be eliminated by moving the proxy implementation into kernel space. Figure 13 depicts the total processing time and copying overheads as the number of connections varies from 8 to 64. As expected, the per-packet copying overhead is independent of the number of connections. So is the overhead of rewriting headers for a given packet. As the number of connections grows, the number of rewrite rules grow in proportion, so the overhead of matching a packet to a rule grows slightly, as shown by the slight increase in the total processing overhead; this total overhead grows from 3.485 to 3.976 ms. Note that our implementation uses a naïve linear rule matching algorithm and this overhead can be reduced substantially using more efficient techniques such as those used in routers to match ACLs.

8.5.1 Result

The dominant overhead of our proxy is due to data copying between kernel and user-space, with relatively efficient per-packet header rewrites and rule matching.

8.6 VMShadow case study

Lastly, we evaluate and show the work progress of VMShadow in fingerprinting and assigning latency-sensitive scores, and using WAN live migrations in resolving complex scenarios for improving the VDs' performance. The series of migration are depicted in Fig. 14.

In this experiment, we consider three different types of applications, i.e., local video, text editing and online streaming running inside four identical VMs. For experimental purpose, we constrain US-East and US-West

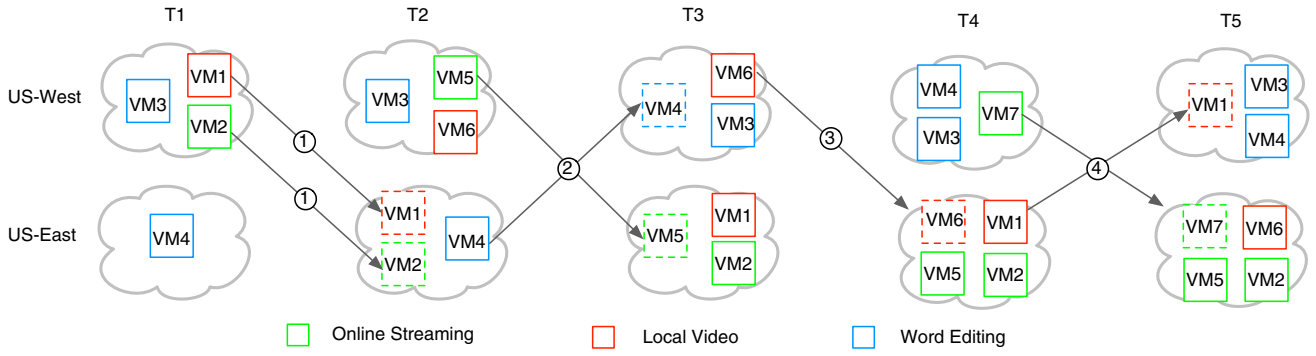


Fig. 14 Illustration of a series of migrations to improve the performance of Desktop VMs. We consider a simplified scenario with two cloud locations, one is closer and the other is further to our thin clients in Massachusetts. VMShadow automatically identifies and prioritizes latency-sensitive desktop VMs, i.e., VDs that run local video and online streaming applications, and migrates them to the US-East cloud location. To accommodate latency-sensitive VMs in a resource-constrained cloud location, VMShadow reclaims the resource by migrating non latency-sensitive VDs to further cloud

sites to both have a capacity of hosting 4 VMs each. Initially only the word editing VM is located at US-East, while the other three are located in US-West. At time T_1, VM_1 and VM_2 with local video and online streaming are ranked high as latency-sensitive and VMShadow triggers their migrations to closer Virginia cloud site. At time T_2, two new desktop VMs, i.e., VM_5 and VM_6 , running video applications are requested and started in Oregon data center. Both of these VMs are also flagged as latency-sensitive and VM_5 is assigned higher latency-sensitive score. To accommodate both of these two VMs in the Virginia data center (currently only has capacity for one more VM), VMShadow first migrates higher rank VM_5 while at the same time reclaims resource by moving lower ranked VM_4 running text editing from Virginia to Oregon. At time T_3 after VM_4 has been successfully migrated, VM-Shadow then continues the process of migrating VM_6. At time T_4, we repeat the event of requesting a new virtual desktop for the user to watch a video streamed from YouTube. This leads to another swap between the newly requested online streaming VM_7 in US-West and the slightly lower ranked VM_1 in US-East. Eventually at time T_5, we end up having all the highly ranked desktop VMs running close to their end-users on the east coast, with lower ranked VMs running in US-West.

Figure 15 depicts the VNC response time for the three desktop VMs running different applications before, during and after their migrations in the above scenario. As shown, the first two VMs have latency-sensitive video activities, and the VNC performance improves significantly after a migration to the US east coast (from 300 to 41.7 ms). The third VM has document editing activity, which does not suffer noticeably despite a reclamation and a migration to west coast, which is further away to its user.

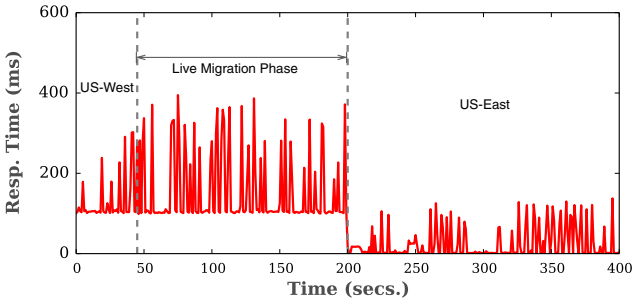
8.6.1 Results

In this case study, we demonstrate VMShadow's ability to discriminate between latency-sensitive and latency-insensitive desktop VMs and to trigger appropriate WAN migrations to improve VNC response time in an artificially constrained cloud environments.

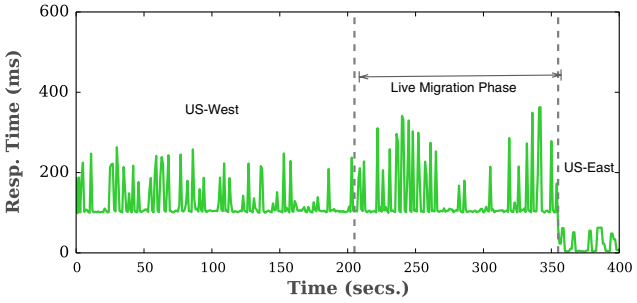
9 Related work

The problem of placing VMs in data centers has been extensively studied, most often as optimization problems, e.g., energy consumption minimization [18–21], or performance maximization [22–24]. However, much of the focus has been, and continues to be, on placing VMs within a single data center. Approaches include devising heuristic algorithms [25, 26] or even formulating placement as a multi-resource bin packing problem [27–29]. Others [30, 31] have even proposed placement and migration approaches that minimize data transfer time within a data center.

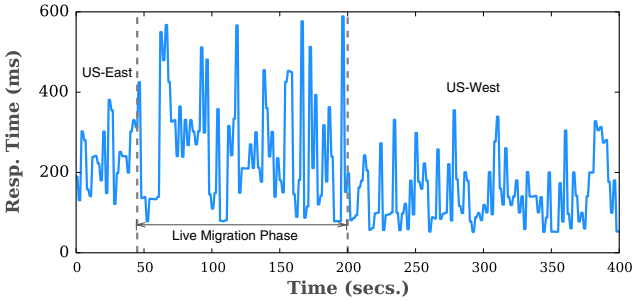
Placement of VMs in a distributed cloud [32, 33] is complicated by additional constraints such as the inter-data center communication cost [34–38]. For example, Steiner et al. [38] demonstrate the challenges of distributing VMs in a distributed cloud using virtual desktop as an example application. There have been a few recent efforts aimed at addressing placement in the distributed cloud [23, 39, 40]. These approaches aim to optimize placement using approximation algorithms that minimize costs and latency [39], or through greedy algorithms that minimize costs and migration time [40–42]. In this work, we dynamically place desktop VMs according to their latency-sensitivities. We seek to balance the performance benefit with the migration cost by taking multiple dimensions into account, including the



(a) Migration of VM_1 that ran Internet Streaming application.



(b) Migration of VM_2 that ran local video application.



(c) Migration of VM_3 that ran local word editor.

Fig. 15 Performance case study of migrating different latency-sensitive VMs. Decisions are made to migrate VM_1 and VM_2 to US-East, to be closer to user. When US-East is resource-constrained, low-ranked VM_3's resources are reclaimed by migrating it back to US-West to free up resources for latency-sensitive VM_2

virtual desktop user behavior, traffic profiles, data center locations and resource availabilities.

The latency-sensitivity of an application is crucial in determining its placement. There has been prior work that evaluated the efficiency of thin-client computing over the WAN and showed that network latency is a dominating factor affecting performance [43, 44]. More recently, [45, 46] propose per-user models that capture the usage profiles of users to determine placement of the front- and back-ends of a desktop cloud.

The ability to manipulate the VM locations agilely, either by cloning [47, 48] or migrating, is the primitive that allows us to adapt to changing latency-sensitivity of VMs. Virtualization platforms provide mechanisms and implementations to achieve LAN live migration with minimal disruption [8, 49]. Multiple efforts [50–53] have also sought to improve efficiency by either minimizing the amount of data transferred [52, 54] or optimizing the number of times data was iteratively transferred [50].

Disruption-free WAN live migration [4, 5, 55, 56] is challenging due to lower wide area bandwidths, larger latencies, and changing IP addresses. Moreover, different cloud locations can run different virtualization platforms. Xen-Blanket [5] provides a thin layer on top of Xen to homogenize diverse cloud infrastructures. CloudNet [4] proposed multiple optimization techniques to dramatically reduce the live migration downtime over the WAN. It also tried to solve the problem of changing IP addresses by advocating "network virtualization" that involved network routers.

Others [57] have suggested using Mobile IPv6 to reroute packets to the new destination. There have also been several proposals [13–16] that attempt to address the general problem of seamless handover of TCP connections across IP address changes. In general all these approaches require changes; either to the applications, the network, or both. In our work, we implement a prototype of VMShadow in Xen by reusing some ideas from CloudNet [4] and Xen-Blanket [5] and use a light-weight connection migration proxy that rewrites packet headers to cope with IP address changes and also to penetrate NATs.

10 Conclusions and future work

In this paper, we presented VMShadow, a system that automatically optimizes the location and performance of VM-based desktops, with dynamic changing needs, running different types of applications. VMShadow performs black-box fingerprinting of a desktop VM's network traffic to infer latency-sensitivity and employs a greedy heuristic based algorithm to move highly latency-sensitive desktop VMs to cloud sites that are closer to their end-users. We empirically showed that desktop VMs with multimedia applications are likely to see the greatest benefits from such location-based optimizations in the distributed cloud infrastructure. VMShadow employs WAN-based live migration and a *new* network connection migration protocol to ensure that the desktop VM migration and subsequent changes to the VM's network address are transparent to end-users. We implemented a prototype of VMShadow in a nested hypervisor and demonstrated its effectiveness for optimizing the performance of VM-based desktops in our Massachusetts-based private cloud and Amazon's EC2 cloud. Our experiments showed the benefits of our approach for latency-sensitive desktops VMs, e.g., those that are running multimedia applications.

In future work, we plan to study the efficacy of using VMShadow for various virtual desktop applications and for other cloud applications beyond virtual desktops.

Acknowledgements We would like to thank all our reviewers for their comments and suggestions. This research was supported by NSF Grants CNS-1117221, CNS-1345300 and OCI-1032765.

References

1. Guo, T., Gopalakrishnan, V., Ramakrishnan, K., Shenoy, P., Venkataramani, A., Lee, S.: Vmshadow: optimizing the performance of latency-sensitive virtual desktops in distributed clouds. In: Proceedings of the 5th ACM Multimedia Systems Conference, pp. 103–114. ACM (2014)
2. Amazon WorkSpaces. https://aws.amazon.com/workspaces/
3. Microsoft Desktop virtualization. https://www.microsoft.com/en-us/cloud-platform/desktop-virtualization
4. Wood, T., Ramakrishnan, K.K., Shenoy, P., Van der Merwe, J.: CloudNet : dynamic pooling of cloud resources by live WAN migration of virtual machines. In: Proceedings of ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments (VEE) (2011)
5. Williams, D., Jamjoom, H., Weatherspoon, H.: The xen-blanket: virtualize once, run everywhere. In: Proceedings of ACM EuroSys (2012)
6. Response times: the 3 important limits. http://www.nngroup.com/articles/response-times-3-important-limits/
7. Katz-Bassett, E., John, J.P., Krishnamurthy, A., Wetherall, D., Anderson, T., Chawathe, Y.: Towards ip geolocation using delay and topology measurements. In: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, IMC '06, pp. 71–84. ACM, New York, NY, USA (2006). doi:10.1145/1177080.1177090
8. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: Proceedings of USENIX NSDI (2005)
9. Heroku:Cloud Application Platform. https://www.heroku.com/
10. Sharma, P., Lee, S., Guo, T., Irwin, D., Shenoy, P.: Spotcheck: Designing a derivative iaas cloud on the spot market. In: Proceedings of the Tenth European Conference on Computer Systems, EuroSys '15, pp. 16:1–16:15. ACM, New York, NY, USA (2015). doi:10.1145/2741948.2741953
11. Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., Warfield, A.: Remus: high availability via asynchronous virtual machine replication. In: Proceedings of USENIX NSDI (2008)
12. Aggarwal, B., Akella, A., Anand, A., Balachandran, A., Chitnis, P., Muthukrishnan, C., Ramjee, R., Varghese, G.: Endre: an end-system redundancy elimination service for enterprises. In: Proceedings of USENIX NSDI (2010)
13. Host Identity Protocol (HIP). http://tools.ietf.org/html/rfc5201
14. Locator/ID Separation Protocol (LISP). http://www.lisp4.net/
15. Identifier-Locator Network Protocol (ILNP). http://tools.ietf.org/html/rfc6740.txt
16. Nordström, E., Shue, D., Gopalan, P., Kiefer, R., Arye, M., Ko, S.Y., Rexford, J., Freedman, M.J.: Serval: An end-host stack for service-centric networking. In: Proceedings of USENIX NSDI (2012)
17. Ford, B., Srisuresh, P., Kegel, D.: Peer-to-peer communication across network address translators. In: Proceedings of USENIX Annual Technical Conference (2005)
18. Dong, J., Jin, X., Wang, H., Li, Y., Zhang, P., Cheng, S.: Energy-saving virtual machine placement in cloud data centers. In: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, pp. 618–624 (2013). Doi:10.1109/CCGrid.2013.107
19. Le, K., Bianchini, R., Zhang, J., Jaluria, Y., Meng, J., Nguyen, T.D.: Reducing electricity cost through virtual machine placement in high performance computing clouds. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 22. ACM (2011)
20. Teng, F., Deng, D., Yu, L., Magouls, F.: An energy-efficient vm placement in cloud datacenter. In: High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on, pp. 173–180 (2014)
21. Wu, G., Tang, M., Tian, Y.C., Li, W.: Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm, pp. 315–323. Springer, Berlin, Heidelberg, Berlin, Heidelberg (2012). doi:10.1007/978-3-642-34487-9_39
22. Do, A.V., Chen, J., Wang, C., Lee, Y.C., Zomaya, A.Y., Zhou, B.B.: Profiling applications for virtual machine placement in clouds. In: Cloud Computing (CLOUD), 2011 IEEE International Conference on, pp. 660–667 (2011). doi:10.1109/CLOUD.2011.75
23. Guo, T., Shenoy, P.: Model-driven geo-elasticity in database clouds. In: Autonomic Computing (ICAC), 2015 IEEE International Conference on, pp. 61–70. IEEE (2015)
24. Jiang, J.W., Lan, T., Ha, S., Chen, M., Chiang, M.: Joint vm placement and routing for data center traffic engineering. In: INFOCOM, 2012 Proceedings IEEE, pp. 2876–2880. IEEE (2012)
25. Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Towards predictable datacenter networks. In: Proceedings of the ACM SIGCOMM (2011)
26. Guo, C., Lu, G., Wang, H.J., Yang, S., Kong, C., Sun, P., Wu, W., Zhang, Y.: Secondnet: a data center network virtualization architecture with bandwidth guarantees. In: Proceedings of ACM CoNEXT (2010)
27. Coffmann, E.G., Gary, M.R., Johnson, D.S.: Approximation algorithms for bin-packing-an updated survey. Algorithm Design for Computer System Design, pp. 49–106 (1984)
28. Mishra, M., Sahoo, A.: On theory of vm placement: anomalies in existing methodologies and their mitigation using a novel vector based approach. In: Cloud Computing (CLOUD), 2011 IEEE International Conference on, pp. 275–282. IEEE (2011)
29. Xu, J., Fortes, J.A.: Multi-objective virtual machine placement in virtualized data center environments. In: Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on and Int'l Conference on Cyber, Physical and Social Computing (CPSCom), pp. 179–188. IEEE (2010)
30. Piao, J.T., Yan, J.: A network-aware virtual machine placement and migration approach in cloud computing. In: Proceedings of Grid and Cooperative Computing (GCC 2010), pp. 87–92 (2010). doi:10.1109/GCC.2010.29
31. Yapicioglu, T., Oktug, S.: A traffic-aware virtual machine placement method for cloud data centers. In: Proceedings of the 2013 IEEE/ACM 6th international conference on utility and cloud computing, pp. 299–301. IEEE Computer Society (2013)
32. Chaisiri, S., Lee, B.S., Niyato, D.: Optimal virtual machine placement across multiple cloud providers. In: Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pp. 103–110. IEEE (2009)
33. Hao, F., Kodialam, M., Lakshman, T.V., Mukherjee, S.: Online allocation of virtual machines in a distributed cloud. In: IEEE INFOCOM 2014—IEEE Conference on Computer Communications, pp. 10–18 (2014). doi:10.1109/INFOCOM.2014.6847919

34. Bronson, N., Amsden, Z., Cabrera, G., Chakka, P., Dimov, P., Ding, H., Ferris, J., Giardullo, A., Kulkarni, S., Li, H., et al.: Tao: Facebooks distributed data store for the social graph. In: Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), pp. 49–60 (2013)
35. Chen, K.y., Xu, Y., Xi, K., Chao, H.J.: Intelligent virtual machine placement for cost efficiency in geo-distributed cloud systems. In: 2013 IEEE International Conference on Communications (ICC), pp. 3498–3503. IEEE (2013)
36. Project Voldemort. http://project-voldemort.com/
37. Sovran, Y., Power, R., Aguilera, M.K., Li, J.: Transactional storage for geo-replicated systems. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 385–400. ACM (2011)
38. Steiner, M., Gaglianello, B.G., Gurbani, V., Hilt, V., Roome, W., Scharf, M., Voith, T.: Network-aware service placement in a distributed cloud environment. In: Proceedings of the ACM SIGCOMM (2012)
39. Alicherry, M., Lakshman, T.V.: Network aware resource allocation in distributed clouds. In: INFOCOM (2012)
40. Guo, T., Sharma, U., Shenoy, P., Wood, T., Sahu, S.: Cost-aware cloud bursting for enterprise applications. ACM Trans. Internet Technol. **13**(3), 10 (2014)
41. Chaisiri, S., Lee, B.S., Niyato, D.: Optimization of resource provisioning cost in cloud computing. IEEE Trans. Serv. Comput. **5**(2), 164–177 (2012)
42. Guo, T., Sharma, U., Wood, T., Sahu, S., Shenoy, P.: Seagull: intelligent cloud bursting for enterprise applications. In: Proceedings of USENIX Annual Technical Conference (2012)
43. Calyam, P., Rajagopalan, S., Selvadhurai, A., Mohan, S., Venkataraman, A., Berryman, A., Ramnath, R.: Leveraging openflow for resource placement of virtual desktop cloud applications. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 311–319. IEEE (2013)
44. Lai, A.M., Nieh, J.: On the performance of wide-area thin-client computing. ACM Trans. Comput. Syst. **24**(2), 175–209 (2006). doi:10.1145/1132026.1132029
45. Abe, Y.: Liberating virtual machines from physical boundaries through execution knowledge (2015)
46. Hiltunen, M., Joshi, K., Schlichting, R., Yamada, N., Moritsu, T.: CloudTops: Latency aware placement of Virtual Desktops institution Distributed Cloud Infrastructures (2013)
47. Lagar-Cavilla, H.A., Tolia, N., De Lara, E., Satyanarayanan, M., OHallaron, D.: Interactive resource-intensive applications made easy. In: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 143–163. Springer (2007)
48. Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., Patchin, P., Rumble, S.M., De Lara, E., Brudno, M., Satyanarayanan, M.: Snowflock: rapid virtual machine cloning for cloud computing. In: Proceedings of the 4th ACM European conference on Computer systems, pp. 1–12. ACM (2009)
49. Nelson, M., Lim, B.H., Hutchins, G.: Fast transparent migration for virtual machines. In: Proceedings of the annual conference on USENIX Annual Technical Conference (2005)
50. Breitgand, D., Kutiel, G., Raz, D.: Cost-aware live migration of services in the cloud. In: Proceedings of Annual Haifa Experimental Systems Conference (2010)
51. Ibrahim, K.Z., Hofmeyr, S., Iancu, C., Roman, E.: Optimized pre-copy live migration for memory intensive applications. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 40. ACM (2011)
52. Jin, H., Deng, L., Wu, S., Shi, X., Pan, X.: Live virtual machine migration with adaptive, memory compression. In: CLUSTER'09, pp. 1–10 (2009)
53. Nathan, S., Bellur, U., Kulkarni, P.: Towards a comprehensive performance model of virtual machine live migration. In: Proceedings of the Sixth ACM Symposium on Cloud Computing, pp. 288–301. ACM (2015)
54. Hou, K.Y., Shin, K.G., Sung, J.L.: Application-assisted live migration of virtual machines with java applications. In: Proceedings of the Tenth European Conference on Computer Systems, p. 15. ACM (2015)
55. Bradford, R., Kotsovinos, E., Feldmann, A., Schiöberg, H.: Live wide-area migration of virtual machines including local persistent state. In: Proceedings of ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments (VEE) (2007)
56. Hirofuchi, T., Ogawa, H., Nakada, H., Itoh, S., Sekiguchi, S.: A live storage migration mechanism over wan for relocatable virtual machine services on clouds. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 460–465. IEEE Computer Society (2009)
57. Harney, E., Goasguen, S., Martin, J., Murphy, M., Westall, M.: The efficacy of live virtual machine migrations over the internet. In: Proceedings of VTDC (2007)