VMShadow: Optimizing The Performance of Latency-sensitive Virtual Desktops in Distributed Clouds

Tian Guo*, Vijay Gopalakrishnan†, K. K. Ramakrishnan†, Prashant Shenoy*,
Arun Venkataramani*, Seungjoon Lee†
*University of Massachusetts Amherst †AT&T Labs - Research
{tian,shenoy, arun}@cs.umass.edu, {gvijay,kkrama,slee}@research.att.com

ABSTRACT

Distributed clouds offer a choice of data center locations to application providers to host their applications. In this paper we consider distributed clouds that host virtual desktops(VDs) which are then accessed by their users through remote desktop protocols. VDs have different sensitivities to latency, primarily determined by the types of applications running (games or video players are more sensitive to latency) and the end users' locations.[1] We design VMShadow, a system to automatically optimize the location and performance of latency-sensitive VDs in the cloud. VMShadow performs black-box fingerprinting of a VM's network traffic to infer its latency-sensitivity and employs a greedy heuristic based algorithm to move highly latency-sensitive VMs to cloud sites that are closer to their end users. VMShadow employs 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 and the public EC2 cloud show that VMShadow is able to discriminate between latency-sensitive and insensitive desktop applications and judiciously move only those VMs that will benefit the most. For desktop VMs with video activity, VMShadow improves VNC's refresh rate by 90%. Further our connection migration proxy, which utilizes dynamic rewriting of packet headers, imposes a rewriting overhead of only 13μs per packet. Trans-continental VM migrations take about 4 minutes.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Design, Management, Measurement, Performance

[1]In the scenario of mobile users, even the network switches between 3G and WiFi affects the latency sensitivity of VDs.

MMSys '14, March 19 - 21 2014, Singapore, Singapore

ACM 978-1-4503-2705-3/14/03 ...$15.00
http://dx.doi.org/10.1145/2557642.2557646 .

Keywords

Distributed Clouds, Virtual Desktop

1. INTRODUCTION

Cloud computing has quickly become a popular paradigm for hosting online applications. Applications ranging from multi-tier web applications to individuals desktops are being hosted out of virtualized resources running in commercial cloud platforms or in a private cloud run by enterprises. This wide range of applications have diverse needs in terms of computation, network bandwidth and latency. To accommodate diverse application needs 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 of nine global locations across four continents. Similarly, enterprise owned private clouds are distributed across a few large data centers and many smaller branch office sites. Such distributed clouds enable an application provider to choose the geographic region(s) that is best suited to the needs of their applications.

A concurrent trend is the growing popularity of 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). The trend which is referred to as virtual desktop infrastructure (VDI) is being adopted in the industry due to numerous benefits. First, virtualizing desktops and hosting them on remote servers simplifies the IT managements tasks such as applying security patches, performing data backups, etc. It also enables better resource management and reduces costs, since multiple desktops VMs can be hosted on a high-end server, which may still be more cost-effective than running each desktop on a separate PC. VDs in the cloud are a growing trend—in addition to their use for business purposes in enterprises, they are also beginning to be offered for consumer use. For instance, commercial services such as *Onlive Desktop* even offer a "free Windows PC in the cloud" that can be accessed from tablets.

The confluence of these trends—the emergence of both distributed clouds and VDs—raises new opportunities and challenges. Today a virtual desktop provider needs to *manually* choose the best data center location for each end-user 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. First, while this may be straightforward in cloud platforms that offer a choice of a few locations (e.g., in 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. This is because regional locations offered by a cloud platform may have smaller hosting capacities than larger "global" locations, implying that there may not be sufficient room to naïvely place all VDs from a region at the nearest regional site. More interestingly, *not all VDs may benefit equally from being placed at the nearest location* to their users—specifically VDs that run latency-sensitive applications such as multi-player games or those that run video playback will see *disproportionately greater benefit* from nearby placement than those that run simple desktop applications such as mail or 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. Further users may themselves move locations, particularly those that may be accessing their VD via mobile devices such as tablets. These challenges imply that static manual placement of VDs at the nearest cloud location may not always be enough or 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 these 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 virtualized desktops in distributed clouds. VMShadow automates the process of placing, monitoring and migrating cloud-based applications across the available cloud sites based on the location of users and latency-sensitivity of the applications. VMShadow performs black-box fingerprinting of a VM's network traffic to infer its latency-sensitivity. It then employs a greedy algorithm that considers the most latency-sensitive applications and uses a cost-benefit metric to choose the VMs that will see the most benefit of migrating to a new location at the lowest cost. VMShadow implements such moves by live migrating the disk and memory state of a VM over a WAN and employs optimizations such as delta encoding and content-based redundancy elimination from prior work [27] to enhance the efficiency of such WAN migrations. Since a migration to a new data center will inevitably change VM's IP address, VMShadow seeks to maintain existing TCP connections between the clients and server VMs through the use of connection proxies. The connection proxies communicate the IP address and port number changes and rewrite the network packet (IP) headers to ensure transparency to the applications. As a result, VMShadow allows for the client to stay connected irrespective of whether the server or even the client moves, whether or not the client or server is behind a NAT, and without requiring the cooperation of network entities such as routers and NAT devices.

Although VMShadow is designed to be a general platform, it is employed to optimize the performance of *desktop clouds* (see Figure 1) in this paper. 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 (Xen-Blanket [26]) and experimentally evaluate its efficacy on a mix of latency-sensitive multimedia and non-latency-sensitive VDs running on a Xen-based private cloud and Amazon's EC2 public cloud. Our results show that VMShadow's black-box fingerprinting approach is able to discriminate between latency-sensitive and -insensitive desktop applications and judiciously moves only those VDs that see the most benefit from being migrated, such as ones with video activity. For example, VDs with video playback activity see up to 90% improvement in refresh rates due to VMShadow's automatic optimizations. We demonstrate live migrations of VDs across Amazon EC2 data centers with trans-coastal VM migrations taking 4 minutes and

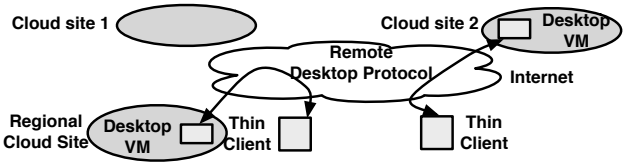


Figure 1: Distributed clouds and desktop clouds.

show that our connection migration proxy, which is based on dynamic rewriting of packet headers, imposes an rewriting overhead of 13μs 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

Consider a cloud computing platform that provides an *infrastructure as a service (IAAS)* to its customers. An IAAS cloud allows application providers to rent servers and storage and to run any virtualized application on top of 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 9 different global locations such as Virginia, Singapore, Tokyo, São Paulo, Sydney etc. 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). Such a distributed cloud is assumed to employ a heterogeneous data center architecture—some locations or sites will comprise very large ("global") data centers, while many other regional sites will comprise smaller data centers as depicted in Figure 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 good economies of scale, while smaller regional sites offer greater choice in placing latency-sensitive application. 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). Each desktop VM represents a "desktop computer" for a particular user who runs traditional desktop applications and stores data on it. Since VDs reside

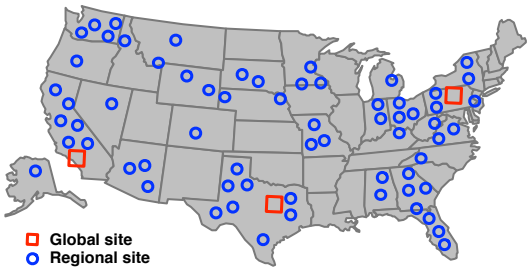


Figure 2: A hypothetical distributed clouds consisting of 3 global cloud sites(red rectangle boxes) and numerous smaller regional sites(blue circles) near major population centers. Content distribution networks have build their data centers in this fashion and we expect distributed clouds to evolve similarly.

in a remote cloud, users connect to their desktop from a thin client using remote desktop protocols such as VNC or Windows RDP. Cloud-based desktops are becoming popular in enterprises, particularly in software "test-and-dev" environments or for users with simple applications, as they eliminate the deployment and management of large numbers of physical machines. 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, despite significant user interactions, not all desktop VMs are equal in terms of being latency-sensitive. As we show in Section 4, the performance of certain desktop applications is significantly impacted by the VD's location relative to its end-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 desktop, 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

Given a distributed cloud, our goal is to design VMShadow, a system that optimizes the performance of cloud-based VDs via judicious placement across different cloud sites. In essence, our system seeks to dynamically map *latency-agnostic* VMs to larger back-end sites for economies of scale and *latency-sensitive* ones to local (or nearby regional) sites for better user experience. To do so, our system must fingerprint a VM's traffic in order to infer its degree of latency-sensitivity while respecting 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) tend to be best candidates for such migrations and we must infer this information while assuming the VM is a black-box with no visibility into the applications running inside the VM. Finally, our system should transparently address networking issues such as IP address change when a VM is moved to a different data center location, even if the client or desktop are behind NATs.

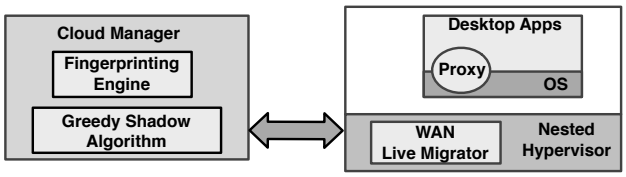


Figure 3: VMShadow Architecture.

	LAN (ms)				US-WEST (ms)			
	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

Table 1: VNC frame response times for latency-insensitive applications.

VMShadow architecture: Figure 3 depicts the high-level architecture of VMShadow. Our system achieves the above goals by implementing four components: (i) a black-box VM fingerprinting technique that infers latency-sensitivity of VMs, (ii) a greedy algorithm that uses cost-benefit metrics to judiciously move highly latency-sensitive VMs at the least cost, (iii) WAN-based live migration of a VM's disk and memory state, with WAN-specific optimizations, and (iv) transparent migration of active TCP connections to ensure seamless connectivity despite IP address changes. The following sections describe the design of each of these components in detail.

4. BLACK-BOX VM FINGERPRINTING

VMShadow uses a black-box fingerprinting technique in order to determine the latency-sensitivity of each VD. The approach is based on the premise that certain applications perform well (or see significant performance improvements) when located close to their users. Consider desktop users that play games; clearly the nearer the VD to the user, the smaller the RTT between the desktop and the user's thin client, and the better the perceived performance for 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 MOOCs or corporate training purposes. Although video playback is not a 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 high bandwidth. As the RTT between the thin client display and the remote VD increases, the performance of video playback suffers (see Figure 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 100ms (typical for trans-US distances), 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/second. In this case, locating the VD closer to the end-user yields a lower RTT and potentially larger refresh rates and better performance. This is depicted in Figure 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. When watching youtube, we observe about 82% of the frame requests of LAN local streaming are served in less than 41.7 ms, which is the update frequency for 24 FPS video. However, when user is watching video on the virtual desktop hosted at US-West about 70% VNC frames are

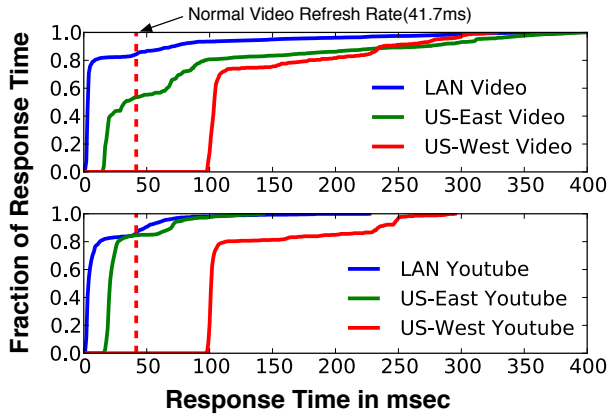


Figure 4: The CDF of VNC frame response times for latency-sensitive applications.

update after more than 125 ms, which indicates the potential loss of video frames. The results are similar when watching a video stored on the local disk of the VM. 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.

In contrast, applications such as simple web browsing and word processing as shown in Table 1 are not very latency-sensitive. Although these are interactive applications, user-perceived performance is not impacted by larger RTT since they are within human *tolerance for interactivity* (as may be seen by growing popularity of cloud-based office applications such as Google docs and Office 360). *Result: Different VDs have different degrees of latency-sensitivity depending on the applications they run.*

VMShadow's black-box fingerprinting involves analyzing a VM's network traffic (e.g., port numbers, traffic characteristics, server addresses) to determine the degree of latency-sensitivity of each VM. Black-box fingerprinting implies this must be done from the outside—by observing packet headers of a VM's incoming and outgoing traffic from the driver domain of the hypervisor (e.g., Xen's dom0). Assume the administrator specifies a list of well-known port numbers for applications that are deemed latency-sensitive[2], VMShadow then periodically samples the VM's network traffic in the hypervisor, aggregates this information at a central fingerprinting engine, and computes a latency-sensitive rank for each VM.

The latency-sensitive rank is computed using three metrics: (i) the fraction of active ports that are flagged to be latency-sensitive—the more the number of latency-sensitive applications within the VM, the greater its rank; (ii) the fraction of latency-sensitive traffic relative to the total network traffic generated by the VM—the greater this fraction, the higher the rank; and (iii) the bandwidth consumed by the remote desktop protocol between the VD and the thin client (higher bandwidth implies a higher rank). Currently the ranking function computes the rank as a weighted combination of the above factors; the rank is updated periodically as an exponential moving average of the current value and recent historical values. 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.

[2]The port numbers list could be refined based on their contributions to latency decrease

5. VMSHADOW ALGORITHM

In this section, we describe the algorithm employed by VMShadow to enable virtual desktop deployments to "shadow" (i.e., follow) their users through intelligent placement. Given a distributed cloud with L locations and N desktop VMs with their current placements and their latency-sensitive ranks, the shadowing algorithm employs the following steps:

Step 1. Identify potential candidates to move: First, VMShadow determines which VMs are good candidates for migration to a different location. Given the latency-sensitive ranks of all VMs, any VMs with a rank above a pre-defined high threshold T_h become a candidate for relocation. Similarly any VM that is placed at a regional site and is ranked below a low threshold T_l (indicating it is no longer very latency-sensitive) becomes a candidate for eviction (i.e., moved back to a larger/global site, since resources at the regional site may be limited). As an example, a desktop VM with consistent new video or gaming activities will become 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 potential candidate: For each VM that is flagged as a candidate for potential migration, 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 [19]). 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, 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 has 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; low-ranked VMs could be evicted when necessary to free up resources.

In our case, this cost-benefit tradeoff can be achieved by formulating the problem as an Integer Linear Program (ILP) optimization and using the output of the solution as a basis of placement decisions. Since an ILP can have exponential running costs, we also devise an efficient greedy heuristic that incorporates this cost-benefit tradeoff. Our greedy algorithm is described in Section 5.1 and we refer the reader to [14] for details of our ILP.

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 adapts the migration optimizations proposed in CloudNet [27] to enable efficient migration of VMs over the WAN. We implement this approach in a nested hypervisor [26] to enable VMShadow to run in public clouds (such as EC2) as well as private clouds. Further, we integrate a connection migration protocol into the VM migration to ensure transparent migrations of active socket connections. Our VM and connection migration techniques are detailed in Section 6.

5.1 The VMShadow Greedy Heuristic

The simplest greedy approach is *rank-ordered greedy*. In this approach, we consider all desktop VMs whose latency-sensitive rank exceeds a certain threshold 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.

An alternate greedy approach is to consider candidates in the order of relative benefit rather than rank. This approach, which we call *cost-oblivious greedy*, considers all VMs that are ranked above a threshold and orders them by the relative benefit 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

$$B = w_1 \cdot (l_1 - l_2) + w_2 \cdot \frac{(l_1 - l_2) * 100}{l_1} \quad (1)$$

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 100ms to 60ms, yielding a 40% reduction, while another may see a decrease from 2ms to 1ms, 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).

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

$$C = (S_{disk} + S_{mem}) \cdot \frac{1}{1 - r} \quad (2)$$

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.[3] 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. Here, for simplicity, we ignore the cost of evicting such low-ranked 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.

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 [9], 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 [27], but with suitable modifications for VMShadow's needs.

[3]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 $(S_{disk} + S_{mem}) \cdot r$, where r is the dirtied fraction. The next round will need an additional $(S_{disk} + S_{mem}) \cdot r^2$. Thus we obtain an expression: $(S_{disk} + S_{mem}) \cdot (1 + r + r^2 + \ldots)$. This expression can be further refined by using different disk and memory dirtying rates for the VM.

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 enterpise 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., if Onlive's Desktop service were implemented on top of Amazon's EC2). 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 [26], which we use in our prototype has shown that this overhead is minimal. As a 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 of 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 [9]. 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 [27] 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, 27]—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% [27].

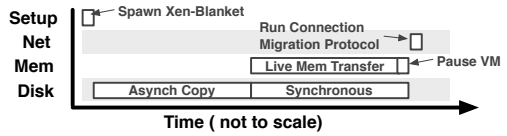


Figure 5: The phases of a migration for non-shared disk, memory and the network in VMShadow using Xen-Blanket.

Step 4: Once the desktop VM moves to a new data center, it typically aquires 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). 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 [1], LISP [3], ILNP [2] and Serval [23] that try to address this problem by separating the host identifier from the network address. 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). The proxy is in the data path for the TCP connection between end points and masks 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 in-

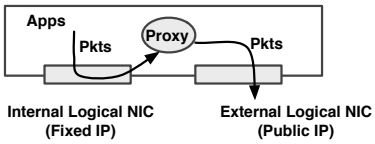


Figure 6: Inside each VM, the proxy bridges an internal logical NIC with the external one, masking the potential IP address changes from the higher-level applications.

terface 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: $(IP_{int}, *) \rightarrow (IP_{ext}, *)$. 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 $(IP_{new}, *) \rightarrow (IP_{old}, *)$ while outgoing packets to the desktop VM see rewrites to the destination IP address as $(*, IP_{old}) \rightarrow (*, IP_{new})$. 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

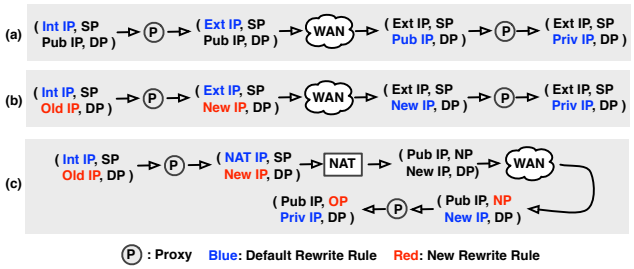


Figure 7: Dynamic packet headers' rewriting sequences. (a) Default rewrite rule is applied when both end points have public IPs. (b) A new rewrite rule is employed when the destination IP address changes. (c) A behind-NAT end point tries to communicate to an entity with new public IP address.

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: $(IP_{int}, srcPort, IP_{old}, dstPort) \rightarrow (IP_{ext}, srcPort, IP_{new}, dstPort)$.

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 [12], 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 peforms the first rewrite $(IP_{int}, srcPort, IP_{old}, dstPort) \rightarrow (IP_{NAT}, srcPort, IP_{new}, dstPort)$. 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 [26]. Our prototype is written in C and Python and consists of several interacting components as shown in Fig. 3. The VM fingerprinting component of VMShadow is implemented 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 to compute a list of active ports that are latency-sensitive (from a pre-configured list of such ports) as well as the volume of traffic on such ports and sends this information to the VMShadow's *cloud manager*. The WAN migration component is implemented inside the nested hypervisor, i.e., the Xen related code in Xen-Blanket, —it modifies the live migration code in Xen to include (i) DRBD-based disk state migration, (ii) rate control mechanisms to control the rate of state transfer over WAN links and (iii) optimizations such as block and page deltas, and content-based redundancy elimination [27] to optimize the data transfers over the WAN.

The connection migration proxy is implemented as a python process 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. Packets are reinserted into the queue once the headers have been rewritten. We implemented the packet header rewriting in user-space mostly as a convenience; the next section measures the overhead of our user-space implementation. A kernel implementation would be more efficient and desirable for production use. Finally, we use the python scapy library to generate appropriate packets for NAT hole punching.

The *Cloud Manager* is a centralized component, also written in python, that (i) computes the latency-sensitivity of each VD using the traffic statistics periodically sampled from each VD, (ii) implements the greedy algorithm to determine where to move which VDs, and (iii) triggers the necessary VM and connection migrations. The Cloud Manager can also limit the number of concurrent live migrations [7] as well as rate control the bandwidth used by each VM migration to limit the impact on foreground traffic and therefore end-users.

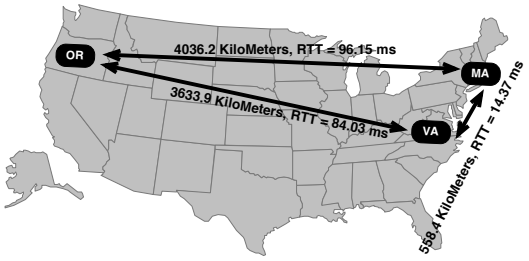


Figure 8: Three cloud sides used for our experiments: a private cloud side in Massachusetts, and EC2 sites in Virginia and Oregon.

8. EXPERIMENTAL EVALUATION

In this section, we describe our experimental setup and then present our experimental results.

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 2.4GHz 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.

Our desktop cloud consists of Ubuntu 12.04 LTS desktops that run a variety of desktop applications, including OpenOffice for editing documents, Firefox browser for web-browsing and watching Youtube video clips, Thunderbird email client and local video playback. Each desktop VM is assigned 1GB memory, 1 VCPU and has a 2GB disk of which 1.32GB is used and runs inside Xen-Blanket dom0. We use VNC as the remote desktop protocol and connect to each desktop using a modified python VNC viewer from laptop-based thin client machines.

8.1 VMShadow Microbenchmarks

Connection Migration Proxy Performance: First 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 2 and Fig 9.

	total time	copy time	rewrite time
Average (ms)	3.375	3.36	0.0133
Std. Dev.	0.022	0.034	0.0042

Table 2: Per-packet proxy overhead of data copying and header rewriting.

The proxy overhead includes (i) data copying overhead incurred by *libnetfilter Queue* in copying packets from kernel-space to user-space and copying back to re-insert packets, (ii) matching a packet to rewrite rules, and (iii) rewriting packet headers. Table 2 depicts the per-packet overhead incurred by the proxy across all runs. As shown in the table, our user-space proxy adds a 3.37ms processing latency to 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 3.36ms overhead of data copying. This overhead can be eliminated by moving the

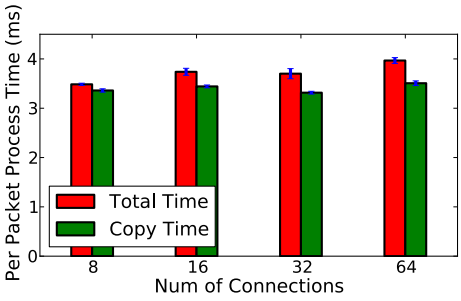


Figure 9: Proxy processing overhead with varying number of active TCP connections.

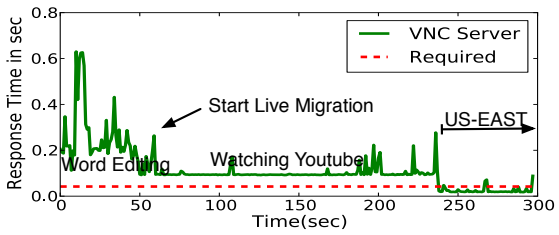
proxy implementation into kernel space. Figure 9 depicts the total processing time and copying overheads as the number of connections varies from 8 to 64. As expected, the per-packet copying overhead is independent of the number of connections. So is the overhead of rewriting headers for a given packet. 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 3.485ms to 3.976ms. Note that our implementation uses a naïve linear rule matching algorithm and this overhead can be reduced substantially by using more efficient techniques such as those used in routers to match ACLs.

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.

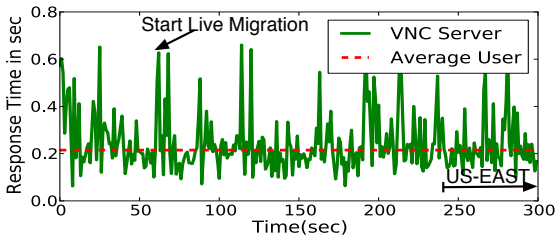
VM Fingerprinting: Our next set of micro-benchmarks focuses on VMShadow's fingerprinting technique. We run a set of applications in different desktop VMs to capture different types of user activities. In each case, we sample the traffic generated by the VM after a warmup period and then fingerprint the VM based on the observed traffic. The sampling of the traffic for fingerprinting is done in the driver domain (dom0) of Xen-Blanket in all cases. Table 3 depicts the behavior of desktop VMs for web-browsing, document editing, watching Youtube and local video playback over 3 mins (a common length of Youtube video clips). 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. Based on this data, and a list of latency-sensitive port numbers, Youtube playback is flagged as latency-sensitive due to the high bandwidth usage and high network traffic percentage. Local video playback also gets a high rank due to higher VNC bandwidth usage (despite not using network traffic). Web browsing and text editing get low ranks as they do not use latency-sensitive ports and due to the low bandwidth usage in those scenarios.

Result: By fingerprinting realistic traffic data of desktop VMs running different applications, we are able to evaluate the ranks of Youtube, local video playback, web browsing and text editing. And such ranks match our hypothesis.

Live Migration and Virtual Desktop Performance: Next, we micro-benchmark the overheads and benefits of WAN-based live migrations implemented by VMShadow. 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 word editing application and watching a YouTube



(a) After migration, online streaming achieves higher VNC frame update frequency due to lower RTT, directly improving user experience.



(b) Word editing application does not see a obvious improvement after migration.

Figure 10: Impact of WAN migration on latency-sensitive and latency-insensitive applications.

	Word+YouTube	Word
Mem (GB)	0.56	0.54
Disk (GB)	1.36	1.34
Total Time(s)	265	249
Pause Time(s)	2.48	2.8

Table 4: The desktop VM WAN migration from OR to VA: data movement and time with VMShadow, including delta-based and CBR optimizations.

video, while the second desktop represents a user only performing word editing. We perform live migration of each VM from US-West to US-East, 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 4, the delta-based and CBR optimizations used by VMShadow allow WAN migrations to be efficient; VMShadow can migrate a desktop VM with 1 GB memory coast-to-coast in about 250 to 265 seconds, depending on the workload. 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 seconds. Figure 10 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 Figure 10(a), 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 100ms RTT between Oregon and Massachusetts, which limits VNC to no more than 10 refreshes per

	To client & other servers				From client & other servers			
	Youtube	Video	Browsing	Text Edit	Youtube	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

Table 3: Fingerprinting Desktop VMs running different applications.

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 the screen at an adequate rate. Figure 10(b) 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 100ms 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.

Results: Migrating a desktop VM trans-continentally takes about 4 mins depending on the workload while incurring 2.5 to 2.8 secs 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.

Greedy Shadow Algorithm: Finally, we micro-benchmark the VMShadow *Cloud Manager* that implements the greedy shadow algorithm for optimizing the location of desktop VMs. We also implemented the integer linear program (ILP) version which provides optimal results but with higher execution time. Our micro-benchmarks 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 40. 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 1200. Figure 11(a) 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(upper plot) and increasing VMs(lower plot); the execution time of the greedy approach, in comparison, remains flat for both scenarios. Also as shown in Figure 11(c), increasing the cloud locations beyond 30 or the number of VMs beyond 800 causes an exponential growth in execution times of the ILP. Figure 11(b) evaluates the effectiveness of the two algorithms in reducing the latency of desktop VMs via migrations. The figure shows that the greedy approach is within 51-56% of the "optimal" ILP approach across various runs. Our results show that 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.

Results: VMShadow's greedy algorithm is able to achieve 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.

8.2 VMShadow Case Study

While our microbechmarks demonstrate the overheads and benefits of our approach for simple scenarios, we now evaluate the VMShadow prototype for more complex scenarios. We experiment with multiple live migrations between the US-East (Virginia) and US-West (Oregon) locations of Amazon EC2. All thin clients are residing at the Massachusetts' cloud location. The series of migrations for improving the VDs' performance is depicted in Figure 12.

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 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, the VMs with local video and online streaming are ranked high as latency-sensitive and VMShadow triggers their migrations to US-East. At time T_2, two new desktop VMs again running video applications are requested and started in US-West. Since both of these VMs are also flagged as latency-sensitive, they are candidates for migration. To accommodate these VMs, VMShadow first reclaims space by moving the lower ranked desktop running text editing from US-East to US-West and then migrates the newly started VMs to US-East. At time T_3, 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 in US-West and the video VM in US-East (since the video activity in that VM subsides, lowering its rank). Eventually at time T_4, we end up having all the highly ranked desktop VMs running close to their end-users on the east coast, with low ranked VMs running in US-West.

Figure 13 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 300ms to 41.7ms). 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.

Results: In this case study, we demonstrate VMShadow's ability to discriminate between latency-sensitive and -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. However, much of the focus has been, and continues to be, on placing VMs within a given data center. Approaches include devising heuristic algorithms [6, 13] or even formulating placement as a multi-resource bin packing problem [10]. Others [24] have even proposed placement and migration approaches that minimize data transfer time within a data center.

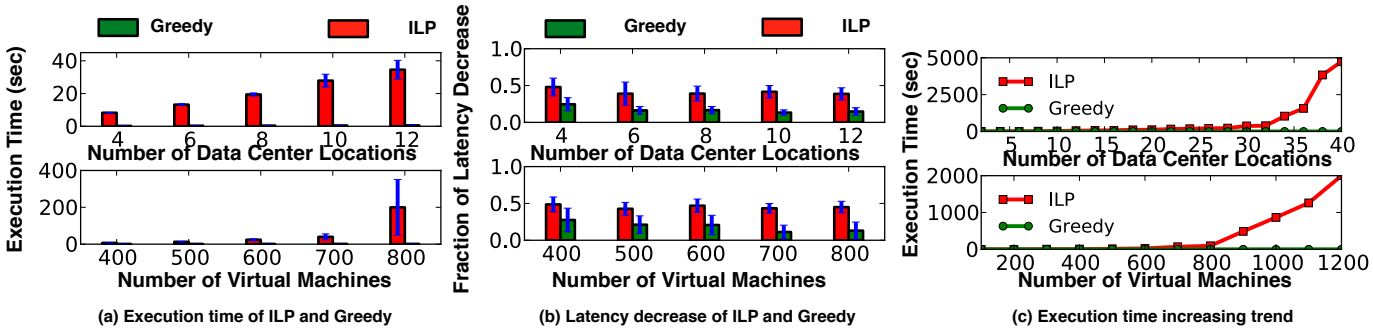


Figure 11: The execution time and difference in accuracy between ILP and greedy algorithms

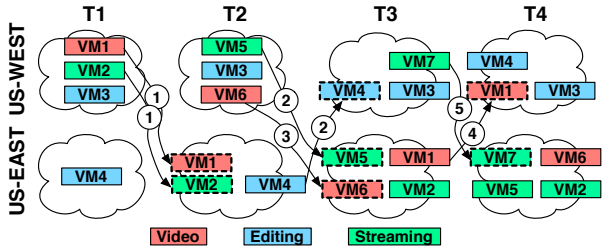


Figure 12: A series of migrations to improve the performance of Desktop VMs.

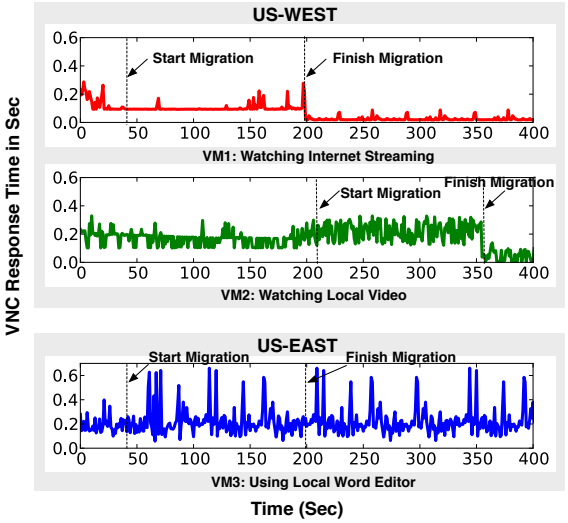


Figure 13: Decisions are made to migrate VM1 and VM2 to US-EAST, to be closer to user. When US-EAST is resource-constrained, low-ranked VM3's resources are reclaimed by migrating it back to US-WEST, after which VM1 and VM2 are migrated to US-East.

Placement of VMs in a distributed cloud is complicated by additional constraints such as the inter-data center communication cost. For example, Steiner et al. [25] 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. These approaches aim to optimize placement using approximation algorithms that minimize costs and latency [5], or through greedy algorithms that minimize costs and migration time [15]. 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 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 [21]. More recently, Hiltunen [17] et al. proposed 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 [20] 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 [9, 22]. Multiple efforts have also sought to improve efficiency by either minimizing the amount of data transferred [18] or optimizing the number of times data was iteratively transferred [8].

Disruption-free WAN live migration is challenging due to lower wide area bandwidths, larger latencies, and changing IP addresses. Moreover the different cloud locations can run different virtualization platforms. Xen-Blanket [26] provides a thin layer on top of Xen to homogenize diverse cloud infrastructures. CloudNet [27] 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 [16] have suggested using Mobile IPv6 to reroute packets to the new destination. There have also been several proposals [1, 3, 2, 23] 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 [27] and XenBlanket [26] 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 desk-

top 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.

Acknowledgement: This research was supported by NSF grants CNS-1117221 and OCI-1032765.

11. REFERENCES

[1] Host Identity Protocol (HIP). `http://tools.ietf.org/html/rfc5201`.

[2] Identifier-Locator Network Protocol (ILNP). `http://tools.ietf.org/html/rfc6740.txt`.

[3] Locator/ID Separation Protocol (LISP). `http://www.lisp4.net/`.

[4] AGGARWAL, B., AKELLA, A., ANAND, A., BALACHANDRAN, A., CHITNIS, P., MUTHUKRISHNAN, C., RAMJEE, R., AND VARGHESE, G. Endre: an end-system redundancy elimination service for enterprises. In *Proceedings of USENIX NSDI* (2010).

[5] ALICHERRY, M., AND LAKSHMAN, T. V. Network aware resource allocation in distributed clouds. In *INFOCOM* (2012).

[6] BALLANI, H., COSTA, P., KARAGIANNIS, T., AND ROWSTRON, A. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM* (2011).

[7] BARKER, S., CHI, Y., MOON, H. J., HACIGÜMÜŞ, H., AND SHENOY, P. "Cut me some slack": latency-aware live migration for databases. In *Proceedings of Conference on Extending Database Technology* (2012).

[8] BREITGAND, D., KUTIEL, G., AND RAZ, D. Cost-aware live migration of services in the cloud. In *Proceedings of Annual Haifa Experimental Systems Conference* (2010).

[9] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *Proceedings of USENIX NSDI* (2005).

[10] COFFMANN, E. G., GARY, M. R., AND JOHNSON, D. S. Approximation algorithms for bin-packing-an updated survey. *Algorithm Design for Computer System Design* (1984), 49–106.

[11] CULLY, B., LEFEBVRE, G., MEYER, D., FEELEY, M., HUTCHINSON, N., AND WARFIELD, A. Remus: high availability via asynchronous virtual machine replication. In *Proceedings of USENIX NSDI* (2008).

[12] FORD, B., SRISURESH, P., AND KEGEL, D. Peer-to-peer communication across network address translators. In *Proceedings of USENIX Annual Technical Conference* (2005).

[13] GUO, C., LU, G., WANG, H. J., YANG, S., KONG, C., SUN, P., WU, W., AND ZHANG, Y. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of ACM CoNEXT* (2010).

[14] GUO, T., GOPALAKRISHNAN, V., RAMAKRISHNAN, K., SHENOY, P., VENKATARAMANI, A., AND LEE, S. Vmshadow: Optimizing the performance of virtual desktops in distributed clouds. Tech. rep., 2013.

[15] GUO, T., SHARMA, U., WOOD, T., SAHU, S., AND SHENOY, P. Seagull: intelligent cloud bursting for enterprise applications. In *Proceedings of USENIX Annual Technical Conference* (2012).

[16] HARNEY, E., GOASGUEN, S., MARTIN, J., MURPHY, M., AND WESTALL, M. The efficacy of live virtual machine migrations over the internet. In *Proceedings of VTDC* (2007).

[17] HILTUNEN, M., JOSHI, K., SCHLICHTING, R., YAMADA, N., AND MORITSU, T. CloudTops: Latency aware placement of Virtual Desktops institution Distributed Cloud Infrastructures.

[18] JIN, H., DENG, L., WU, S., SHI, X., AND PAN, X. Live virtual machine migration with adaptive, memory compression. In *CLUSTER'09* (2009), pp. 1–10.

[19] KATZ-BASSETT, E., JOHN, J. P., KRISHNAMURTHY, A., WETHERALL, D., ANDERSON, T., AND CHAWATHE, Y. Towards ip geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2006), IMC '06, ACM, pp. 71–84.

[20] LAGAR-CAVILLA, H. A., WHITNEY, J. A., SCANNELL, A. M., PATCHIN, P., RUMBLE, S. M., DE LARA, E., BRUDNO, M., AND SATYANARAYANAN, M. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European Conference on Computer Systems* (New York, NY, USA, 2009), EuroSys '09, ACM, pp. 1–12.

[21] LAI, A. M., AND NIEH, J. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst. 24*, 2 (May 2006), 175–209.

[22] NELSON, M., LIM, B.-H., AND HUTCHINS, G. Fast transparent migration for virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (2005).

[23] NORDSTRÖM, E., SHUE, D., GOPALAN, P., KIEFER, R., ARYE, M., KO, S. Y., REXFORD, J., AND FREEDMAN, M. J. Serval: an end-host stack for service-centric networking. In *Proceedings of USENIX NSDI* (2012).

[24] PIAO, J. T., AND YAN, J. A network-aware virtual machine placement and migration approach in cloud computing. In *Proceedings of Grid and Cooperative Computing (GCC 2010)* (2010), pp. 87–92.

[25] STEINER, M., GAGLIANELLO, B. G., GURBANI, V., HILT, V., ROOME, W., SCHARF, M., AND VOITH, T. Network-aware service placement in a distributed cloud environment. In *Proceedings of the ACM SIGCOMM* (2012).

[26] WILLIAMS, D., JAMJOOM, H., AND WEATHERSPOON, H. The xen-blanket: virtualize once, run everywhere. In *Proceedings of ACM EuroSys* (2012).

[27] WOOD, T., RAMAKRISHNAN, K. K., SHENOY, P., AND 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)* (Mar. 2011).