Follow the Sun through the Clouds: Application Migration for Geographically Shifting Workloads

Zhiming Shen Qin Jia Gur-Eyal Sela Ben Rainero
Weijia Song Robbert van Renesse Hakim Weatherspoon
Cornell University

Abstract

Global cloud services have to respond to workloads that shift geographically as a function of time-of-day or in response to special events. While many such services have support for adding nodes in one region and removing nodes in another, we demonstrate that such mechanisms can lead to significant performance degradation. Yet other services do not support application-level migration at all. Live VM migration between availability zones or even across cloud providers would be ideal, but cloud providers do not support this flexible mechanism.

This paper presents the Supercloud, a uniform cloud service that supports live VM migration between data centers of all major public cloud providers. The Supercloud also provides a scheduler that automatically determines when and where to move VMs for optimal performance. We demonstrate that live VM migration can support shifting workloads effectively, with low downtimes and transparently to both services and their clients. The Supercloud also addresses challenges for supporting cross-cloud storage and networking.

Categories and Subject Descriptors D.4.7 [*Organization and Design*]: Distributed systems

Keywords Supercloud, Follow the Sun, Application Migration

1. Introduction

Live VM migration [22] is a technology that enables application migration by transparently migrating a VM to different physical locations. Cloud providers can use this to balance load among physical resources or to offload hardware that needs to be replaced or upgraded. Unfortunately, cloud providers have not exposed this powerful capability to its users. Studies have demonstrated that commercial applications have workloads that show both temporal and spatial variability [19, 26, 51], and even a tiny increase in the latency could cause significant revenue drop [9]. For example, Google reported that a 500ms delay in page load times caused a 20% drop in traffic and revenue [3]. Live migration within an availability zone would have little value to users looking to optimize geographic load shifts and would complicate the task of load balancing to cloud providers. Cloud providers do not support migration across availability zones because they provide only local storage in availability zones, and network routing for migrating VMs would add significant complexity. Supporting migrating VMs between different cloud providers is contrary to competitive concerns and also faces technical challenges such as incompatible hypervisors, isolated networks, and heterogeneous storage.

Many cloud services such as ZooKeeper [31] and Cassandra [20] provide mechanisms to add and remove nodes that can be used to support application-level migration. We present experimental results that show that such mechanisms can cause significant service degradation during migration compared to straightforward VM migration. Yet other cloud services such as MySQL do not support application-level migration—VM migration provides a simple solution.

In this paper, we present the Supercloud, a cloud architecture that enables application migration as a service across different availability zones or cloud providers. The Supercloud provides interfaces to allocate, migrate, and terminate resources such as virtual machines and storage and presents a homogeneous network to tie these resources together. Moreover, the Supercloud provides a scheduling framework for automating placement and migration that each application can customize to optimize metrics such as client-perceived latency. While the Supercloud could be deployed as a public cloud service, in its current form it is intended to be used as a private cloud, and different users create their own instantiations.

SoCC '16, October 05-07, 2016, Santa Clara, CA, USA.
© 2016 ACM. ISBN 978-1-4503-4525-5/16/10…$15.00.
DOI: http://dx.doi.org/10.1145/2987550.2987561

The Supercloud can span across all major public cloud providers such as Amazon EC2, Microsoft Azure, Google Compute Engine, and Rackspace etc., as well as private clouds. Supercloud users have the freedom to re-locate VMs to many data centers across the world, irrespective of owner and without having to implement complex re-configuration and state re-synchronization in their applications. Using the Supercloud, an application can easily offload from an overloaded data center to another one with a different infrastructure. We have been working with the NSF Aristotle Cloud Federation project [12] to integrate cloud resources in different universities for supporting research scientists and engineers.

In implementing support for live application-level VM migration we had to address various challenges relating to heterogeneity, networking, and storage. The Supercloud leverages nested virtualization [48], eliminating the need for VM management support from underlying providers. We designed and implemented a new distributed storage system optimized for wide-area cross-provider VM migration. It decouples providing strong consistency from update propagation, improving latency while reducing overhead. We deploy a high performance Software-Defined Network (SDN) built with Open-vSwitch and VXLAN (Virtual Extended LAN) tunnels crossing cloud boundaries and a Frenetic [25] controller. We also designed and evaluated various solutions to deal with migrating services that use public IP addresses. The Supercloud runs the OpenStack platform and appears to users as a single private OpenStack cloud. A new scheduler monitors workload and automatically determines the optimal location for running services.

In this paper we use migration to change VM placement to reduce client-perceived latency. But the Supercloud can also be used to optimize metrics such as price or availability [33], or as a enabling technology for offloading from a overloaded private data center to a public cloud [32].

This paper makes the following contributions:

- We propose Migration-as-a-Service to support geographically shifting workloads (Section 2);
- We design a scheduling framework for handling application migration automatically (Section 3);
- We provide storage and networking solutions for supporting efficient VM migration (Section 4);
- We evaluate the performance of the Supercloud and demonstrate the benefits of leveraging crosscloud VM migration (Section 5).

2. Application Migration

For a service that has global users in different timezones, it would be ideal if it could "follow the sun," that is, continuously shifting to a location where the majority of users experience the lowest possible latency. This is a challenging task even for distributed applications that can migrate by adding and removing nodes, not to mention legacy applications that cannot be easily scaled dynamically.

Distributed applications typically adopt a distributed consensus or transaction protocol in order to support data replication, distributed locking, distributed transactions, and so on. These protocols are designed to be fault-tolerant. However, adding and removing nodes while tolerating failure is a fundamental and challenging problem—it requires changing the "membership" of the system, thus subsequent requests following the membership change must be processed with a different configuration. In addition to the complexity of reaching agreement on configuration, adding and removing nodes triggers complicated internal state transfer and synchronization protocols, and membership reconfiguration is not transparent to clients. As a result, tolerating failure while providing good performance during reconfiguration is non-trivial.

Because membership reconfiguration is generally considered a rare event, applications use ad hoc mechanisms to achieve it with unpredictable performance. For example, MongoDB does not allow changing the sharding key on the fly, and the whole database must be exported and then imported again to change key distribution. As another example, re-configuring a cluster running an old version of ZooKeeper (without the new dynamic reconfiguration feature) requires a "rolling restart"—a procedure whereby servers are shutdown and restarted in a particular order so that any quorum of currently running servers includes at least one server with the latest state [41]. If not performed correctly, one server with outdated state might be elected as the new leader and the whole cluster might enter an inconsistent state. While ZooKeeper recently added support for dynamic reconfiguration, it is inefficient for geographic migration (see Section 5.2.2). A key-value store such as Cassandra can easily add and remove a node and adjust token distribution, but doing these for geographic server migration triggers unnecessary data replication and load re-balancing, and can affect service availability if a failure occurs at the same time (see Section 5.2.1).

In order to "follow the sun", applications must be migrated several times a day, and, as we have discussed, implementing migration by adding and removing nodes is suboptimal. An alternative approach is to use live VM migration since it can be performed transparently to the application and even to clients. For example, if live VM migration were supported, we can migrate the ZooKeeper leader and servers to locations where the leader and a majority of servers are geographically close to most active users, without the need of changing a single line of source code in ZooKeeper. Often this only involves migrating the leader: If the majority of clients reside in two different regions like the US and Asia, and there are $2f+1$ servers, including one leader, then with f servers running in one

location, the US, and f servers running in another, Asia, only the leader needs to migrate back and forth once a day. The downtime due to live migration of a VM between the U.S. and Asia is less than one second, while the total migration time for 1GB memory (which proceeds in the background) is about 100 seconds. The downtime is small enough such that TCP connections do not break nor cause broken sessions or leader re-election.

3. Scheduling Framework

Applications in the Supercloud can make migration decisions by themselves and issue migration commands through the OpenStack API. To facilitate the process of deciding optimal placement, the Supercloud provides a scheduling framework for the user. Users can customize the scheduling policy for different applications by implementing some interfaces. The Supercloud scheduler periodically evaluates current placement of the application and automatically adjusts it when a better placement is found.

Suppose the Supercloud is deployed to N different data centers: $d_1, d_2, ..., d_N$. We denote a placement plan for an application with k nodes as $P = \{p_1, p_2, ..., p_k\}$, where p_i is the location of the i^{th} node. Periodically, the Supercloud measures end-to-end latency between all different data centers and stores the results in a latency matrix L, where $L(i, j)$ is the round-trip-time (RTT) from d_i to d_j. The workload of the application is also captured periodically in a workload statistics report S. S is application-specific, so it should be monitored in the application-level and passed to the scheduling framework.

In order to evaluate a placement plan, the application has to provide 1) an evaluation function $f(P,S,L)$ that evaluates a placement plan under a certain workload and returns a score, and 2) a threshold T that specifies the minimal score change that can trigger VM migration. Using f, the scheduler iterates through all possible placement plans and gets a set of candidate placement plans D that maximize the score and outperform the current placement plan $P_{current}$ by at least T, that is:

$$D = \arg\max_{P} \{ f(P,S,L) | f(P,S,L) \geq f(P_{current},S,L) + T \}$$

To choose a placement plan in D, the scheduler compares each placement plan with the current placement $P_{current}$ and selects the one that requires the fewest migrations.

Below we present case studies to demonstrate policies for two different types of applications.

3.1 Non-Distributed Applications

For a single VM running a non-distributed application such as a MySQL database, a placement plan is simply the location of the VM: $P = \{p\}$. We can deploy a set of service front-ends located in different data centers to collect user requests and forward them to the VM. This architecture makes the location of the VM transparent to clients and can help with simplifying placement evaluation. We will describe the networking architecture in Section 4.2.

The goal of placement is to minimize average latency for all front-ends. Here we only consider latency in the network and ignore processing time for different types of requests. S is defined as:

$$S = \{(s_1, d_{s_1}), (s_2, d_{s_2}), ..., (s_n, d_{s_n})\}$$

where n is the number of front-ends, s_i is the number of active clients of the i^{th} front-end, and d_{s_i} is its location.

Each front-end is assigned a weight, which equals the number of requests it receives. The score of a placement plan is the weighted average latency of all front-ends (negated so lower latencies result in higher scores), that is,

$$f(P,S,L) = -\sum_{i=1}^{n} s_i \cdot L(d_{s_i}, p)$$

3.2 Distributed Applications with Replicated State

Distributed applications typically maintain replicated state using some consensus protocol. Below we use ZooKeeper as an example. ZooKeeper is implemented using a replicated *ensemble* of servers kept consistent using Zab (ZooKeeper Atomic Broadcast) [34]. In Zab, one server acts as a leader that communicates with a majority of servers to agree on a total order of updates. Read requests are handled by any node in the ensemble, while write requests must be broadcast by the leader and agreed upon by the majority of the ensemble.

For a ZooKeeper ensemble with m nodes, a placement plan is the location of all nodes: $P = \{p_1, p_2, ..., p_m\}$, where p_l is the location of the leader. Again we assume that there are n front-ends in different data centers to collect and forward client requests. To evaluate a placement plan, we need to consider read and write requests separately. So:

$$S = \{(r_1, w_1, d_{s_1}), (r_2, w_2, d_{s_2}), ..., (r_n, w_n, d_{s_n})\}$$

r_i and w_i are the number of read and write requests received by the i^{th} front-end and d_{s_i} is its location.

The goal of placement is again to minimize average network latency for all front-ends, ignoring request processing time and only considering network delay. For the purpose of load balancing, ZooKeeper clients are typically connected randomly to one node in the ensemble. Read requests can return immediately. For the i^{th} front-end, the expected read latency is

$$R_i = \operatorname*{avg}_{j=1...m} L(d_{s_i}, p_j)$$

Write requests are processed in three steps:

- Step 1: a write request from the i^{th} front-end goes randomly to one of the ZooKeeper nodes. The average latency is:

$$W_i^{(1)} = \operatorname*{avg}_{j=1...m} L(d_{s_i}, p_j)$$

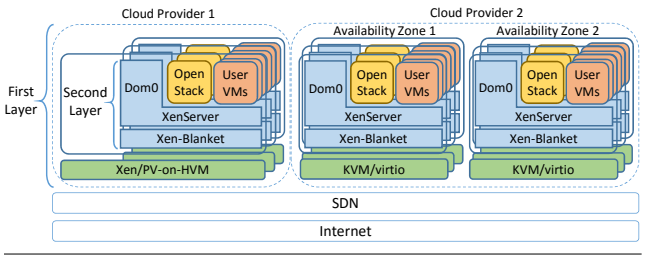


Figure 1. Example deployment of the Supercloud.

- Step 2: the write request is then forwarded to the leader of the ensemble. The average latency for this step is:

$$W_i^{(2)} = \underset{j=1...m}{\text{avg}} \; L(p_j, p_l)$$

- Step 3: the leader broadcasts the request twice in a protocol similar to two-phase commit, and for each broadcast it must wait until at least half of the ensemble replies. The average latency for this step is:

$$W_i^{(3)} = 2 \times \underset{j=1...m, j \neq l}{\text{median}} \; L(p_l, p_j)$$

So the expected network latency for a write request from the i^{th} front-end is: $W_i = W_i^{(1)} + W_i^{(2)} + W_i^{(3)}$. The evaluation function for ZooKeeper is calculating the weighted average network latency of all requests, assuming that we give read and write requests a weight α and β respectively:

$$f(P,S,L) = -\sum_{i=1}^{n} (\alpha \cdot R_i \cdot r_i + \beta \cdot W_i \cdot w_i)$$

4. Supercloud Architecture

We have implemented support for live migration within the context of the Supercloud. The Supercloud can span multiple availability zones of the same provider as well as availability zones of multiple cloud providers and private clusters (see Figure 1). To accomplish this, there are two layers of hardware virtualization. The bottom layer, called *first-layer*, is the infrastructure managed by a Infrastructure as a Service (IaaS) cloud provider such as Amazon EC2 or Rackspace, or managed privately. It provides VMs, cloud storage, and networking. Another layer of virtualization on top of this, called *second-layer*, is the IaaS infrastructure managed by the Supercloud. It leverages resources from the first-layer and provides a single uniform virtual cloud interface. Importantly, the second-layer is completely controlled by users.

In case of compute resources, the first-layer has a hypervisor managed by the underlying cloud provider and a collection of hardware virtual machines (HVMs). We refer to these as *first-layer hypervisors* and *first-layer VMs*.

The second layer, exposed to Supercloud users, is similarly separated into a hypervisor and some number of guest VMs that we call the *second-layer hypervisor* and *VMs*[1]. We use Xen-Blanket [48] for the second-layer hypervisor. Xen-Blanket provides a consistent Xen-based para-virtualized (PV) interface. In Xen, one VM is called *Domain-0* (aka *Dom0*) and manages the other VMs, called *Domain-U*s (aka *DomU*s). A second-layer Dom0 multiplexes resources such as I/O amongst the DomUs.

We use OpenStack [5] to manage user VMs and provide compatibility to existing applications. In particular, a XenServer runs within the second-layer Dom0 and allows OpenStack to manage all second-layer DomU VMs.

A common practice in VM migration is using shared storage to serve VM images, so that a VM migration only needs to transfer the memory. This is essential to good performance because migrating a VM with the disk image takes a long time. XenServer offers two options for sharing storage: an NFS-based solution and an iSCSI-based solution. Both these approaches use a centralized storage repository. While simple, this can lead to significant latencies, low bandwidth, and high Internet cost for VMs that access the disk through a wide area network when migrated to another region or cloud. Previous works have proposed different mechanisms and optimization for wide-area VM migration with disk images [17, 36, 40]. However, migrating the whole image file incurs significant Internet traffic, which is typically charged by cloud providers. To support efficient VM migration in the wide area network, we developed a geo-replicated image file storage that seeks a good balance between performance and cost.

To provide the illusion of a single virtual cloud, the control services (including XenServer and OpenStack) and user VMs need to communicate in a consistent manner no matter where the end-point VMs reside. A migrated user VM expects its IP address to remain unchanged. To accomplish this, the Supercloud network layer is built using a Software-Defined Network (SDN) overlay based on Open vSwitch, VXLAN tunnels, and the Frenetic SDN controller [25]. Such an overlay network gives control over routing for the second-layer and enables compatibility with heterogeneous first-layer networks.

We support all major hypervisors including Xen, KVM, Hyper-V, and VMWare, so a Supercloud instance can span all major cloud providers including Amazon EC2, Rackspace, Windows Azure, Google Compute Engine, and VMWare vCloud Air.

Below we describe storage and networking in more detail.

4.1 Storage

4.1.1 Consistency and Data Propagation

Geo-replicated storage solutions sometimes adopt a weaker consistency model, such as *eventual consistency*, in which applications reading different replicas may see stale results.

[1] User VMs, user VM instances, second-layer VMs, and second-layer DomU VMs are used interchangeably.

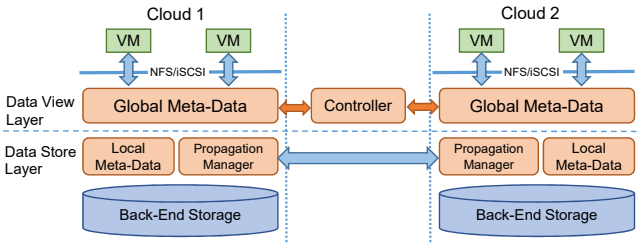


Figure 2. Storage architecture of the Supercloud.

This is not suitable for an image storage on which a migrated VM expects up-to-date data. Using a strongly consistent geo-replicated storage requires synchronous data propagation, resulting in low write throughput. A key observation is that a running application typically does not require all data in the image. Our storage system thus decouples consistency from data propagation.

As shown in Figure 2, the storage service consists of two layers: a data view layer provides a required consistency guarantee, and a data store layer stores and propagates data. Images are divided into blocks with a constant size (4KB). The global meta-data in the data view layer includes a version number for each block. When a VM is reading a block, the version number is compared to the version number in the local meta-data in the local data store. If the latest version of the block is available locally, data is returned immediately. Otherwise, the data store checks the location of the block in the global meta-data and fetches the data remotely. Updating a block increases its version number by one, and the updated global meta-data is then propagated to other replicas.

The consistency model seen by applications is completely determined by the data view layer. Since we only have global meta-data in this layer, it is relatively cheap to implement strong consistency. Since the data store layer is decoupled, data propagation can be optimized separately without concern about consistency.

Our current design optimizes performance and minimizes traffic cost. The Back-End Storage can tolerate failures within a single cloud. However, after several migrations the image may have the latest version of blocks scattered in different clouds, and a cloud-level failure before an updated block is propagated may affect the availability of the whole image. If an image is really critical and would like to tolerate cloud failures, it is possible to pass a hint to the propagation manager so that each write is synchronously propagated to different clouds, at significant cost to application performance.

4.1.2 Global Meta-Data Propagation

Due to the long latency in the wide area network, meta-data transfer affects application performance significantly if this propagation is in the read/write critical path. We make two key observations:

- If an image file is open for writing, it can only be accessed by a single VM. This happens when the image is private to a VM.
- If an image file is shared by multiple VMs, it is read-only. This happens when the image is a snapshot or a base image file.

These observations indicate that, although a migrated VM is expecting strong consistency of the image storage, multiple replicas of the same image file do not need to be identical all the time. It suffices to provide VMs with a "close-to-open" consistency: after a file is closed, reads after subsequent opens will see the latest version. We remove the global meta-data propagation from the read/write critical path by committing the meta-data update locally, and only flushing the global meta-data to a centralized controller when closing the image file. Subsequent opens of the file need to sync the global meta-data with the controller first. Note that the controller is involved only when opening or closing the image file.

4.1.3 Data Propagation Policies

By decoupling the data view and data store layer, a VM can see a consistent disk image no matter where it is migrated. However, fetching each block on-demand through the wide area network significantly degrades read performance. It is useful to proactively propagate a block before migration if we can predict that it is going to be accessed in another place. Intuitively, a block that is read frequently and updated rarely should be aggressively propagated. On the other hand, propagating a block that is updated frequently is a waste of network resources. Because Internet traffic is typically charged to the user, we need to be careful about proactive propagation.

The data store layer monitors the access pattern of the VM and selectively propagates those blocks that are most likely to be accessed after migration and least likely to be updated after propagation. To this end, we maintain a priority queue of updated blocks for each image file. The priority of each block is updated on each read or write. We use the read/write ratio $F = f_r/f_w$ to calculate the priority of propagating a block, where f_r and f_w are the read and write frequency of a block respectively. When two blocks have the same value of F, we first propagate the block with a higher read frequency. So the formula of calculating the propagation priority P^b for a block b is:

$$P^b = \begin{cases} K \cdot f_r^b / f_w^b + f_r^b & \text{if } f_r^b / f_w^b \geq S \\ -1 & \text{if } f_w^b = 0 \text{ or } f_r^b / f_w^b < S \end{cases}$$

The value K determines how much we want to favor the read/write ratio. In our prototype we use $K = 1000$. The constant S determines the aggressiveness of the propagation. The lower S is, the more data will be propagated. When $f_w^b = 0$, which means the block has never been updated, or

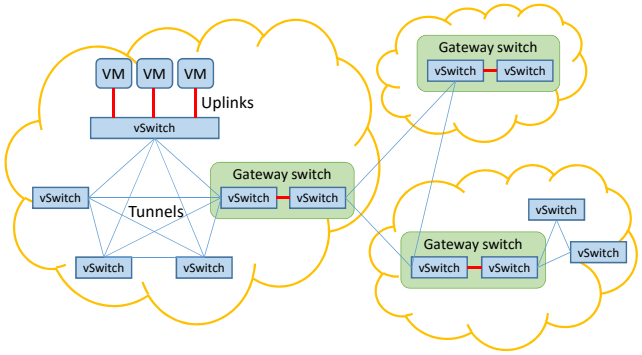


Figure 3. Network topology of the Supercloud.

when $f_r^b / f_w^b < S$, we set the priority to -1 to indicate that this block is not to be propagated proactively. In our current prototype, S is set to 1. The parameters K and S can be tuned separately for each VM, and adjusted on the fly if the workload changes.

After deciding which block to propagate, the next question is where to propagate the block. Since the Supercloud can be deployed to many places even across clouds, propagating each block to all destinations wastes Internet traffic and money. To further optimize data propagation, a transition probability table is added into the image file's meta-data, indicating the probability that a VM is moved from one place to another. Each data block is randomly propagated according to the probability in the transition table, so that the destination to which the VM is most likely to be migrated will receive most propagated blocks. Note that the transition table is a hint given by the user when creating the image file. It is also possible to train the table on the fly if the VM is migrated many times.

4.2 Networking

4.2.1 High-Performance VPN

To enable communication between control services (including XenServer and OpenStack services) and VMs, we place them into a virtual private network (VPN). Good performance requires minimizing the number of hops. Existing VPN solutions such as OpenVPN [6] use a centralized server to forward traffic, which causes high latency and poor throughput. The *tinc* VPN [44] implements an automatic full mesh peer-to-peer routing protocol, minimizing the number of hops traversed between endpoints. However, we found that tinc imposes high performance overhead, mostly caused by extra data-copy and kernel/user mode switching.

To build a high-performance VPN solution for the Supercloud, we use Open vSwitch [4], VXLAN tunnels, and the Frenetic SDN controller [25]. Open vSwitch implements data-paths in kernel mode and supports an OpenFlow-based control plane. Each virtual switch uses an uplink to connect to the VMs running on the same first-layer VM and a set of VXLAN tunnels to connect to all other switches, as illustrated in Figure 3. We create a full mesh network here because we want to always forward packets directly to their destinations. We use VXLAN over GRE (Generic Routing Encapsulation) tunnels because this approach is based on UDP instead of a proprietary protocol, and thus better supported by different firewalls.

In a hierarchical topology, switches running in a private network cannot set up VXLAN tunnels directly with other switches outside the network. A gateway switch is required to forward packets in this case (see Figure 3). The node running the gateway switch is in more than one network. To implement the gateway switch, we create one switch for each of the networks, inter-connected with an in-kernel *patch port*. Each switch builds full mesh connections with other switches in its own network as before and treats the patch port as an uplink.

Switches connected in a full mesh form loops. Ordinarily one would run a spanning tree protocol, but with a network topology demonstrated in Figure 3, a spanning tree cannot minimize the number of hops for every pair of nodes. To route packets efficiently, switches in the Supercloud VPN are connected to a centralized SDN controller implemented with Frenetic [25]. The controller learns the topology of the network by instructing the switches to send a "spoof packet." On receiving the spoof packet, switches report to the controller on which port the packet is received, so that the controller can record how switches are connected. The controller implements a MAC-learning functionality for each switch. The MAC address, IP address, and location of a VM is learned when it sends out packets.

ARP (Address Resolution Protocol) are forwarded directly to the destination instead of broadcast. When routing a packet, the controller calculates the shortest path on the network and installs OpenFlow rules along the path to enable the communication. This is done only once for each flow and subsequent data transportation does not need to involve the controller anymore.

4.2.2 Supporting VM Live-Migration

Supporting VM live migration is another challenge. To keep the IP address of a migrated VM unchanged, the underlying VPN needs to adjust the routing path and re-direct traffic. However, the adjustment cannot be done immediately when the migration is triggered, because at this point the VM is still running in the original location and existing network flows should not be affected.

In order to know at which point the routing path should be adjusted without adding a hook into the hypervisor, before triggering the migration, the controller is notified with the source and destination of the migrated VM. The controller then injects a preparation rule in the destination switch so that it can get an immediate report when a packet with the migrated VM's MAC address is received. When migration is finished, the migrated VM will send out an ARP notification.

This is captured by the controller so that it knows that the migration has finished. The controller then updates all switches that have the migrated VM's MAC address in the MAC table, avoiding the usual ARP broadcast.

4.2.3 Supporting Public IP Addresses

A public IP address is required to expose a service to the public network. We currently support addressing a second-layer VM from the public network by allocating a public IP address to a first-layer VM in the Supercloud network (i.e., a public IP front-end), and then applying port forwarding to map certain ports to the second-layer VM. This solution has good performance because packets can be routed in the public network to the VM directly.

A challenge arises when this VM is migrated to a different cloud provider. Without specific support from Internet Service Providers (ISPs) such as anycast, mobile IP, or multihoming, traffic sent to the public IP address needs to be re-directed by the same first-layer VM, no matter where the second-layer VM currently resides. While this works, it can lead to high latency.

To address this issue, we adopt an idea from Content Distribution Networks (CDN): instead of giving the same public IP address to clients all over the world, we give each client the public IP address of a front-end server in a nearby data center. Taking this extra but short hop, the routing from the front-end to the second-layer VM is always optimized in the public network, performing much better for most clients than a centralized public IP solution.

For applications that do not want to pay the additional cost of the front-end servers, multiple public IP addresses, and an additional hop, the Supercloud has its own dynamic DNS service and can update the DNS mapping after migration. Note that clients might see an out-of-date public IP address before the DNS cache is expired. For services that use protocols such as HTTP, SOAP, and REST, we deploy an HTTP redirection service on the original public IP front-end and respond to clients with an HTTP redirection response.

4.3 Discussion

The benefits of the Supercloud do not come without costs. For example, nested virtualization imposes performance overhead including CPU scheduling delay and I/O overhead. Users can evaluate this tradeoff based on migration frequency and performance requirements, and it is application-dependent. We are currently in the process of building support for containers into the Supercloud. Container technology [42] provides another way to homogenize different cloud platforms. However, compared to live VM migration, which is mature and widely used, container migration [38] technology is preliminary and involves a checkpointing/resume mechanism that might cause a relatively large performance hiccup. Even with mature container migration, the challenges of building a well-performing Supercloud remain essentially the same.

5. Evaluation

The Supercloud represents a new and unique capability: A distributed service can migrate live, and incrementally or whole, between availability zones and heterogeneous cloud providers. In this section, we investigate three research questions enabled by and enabling this capability:

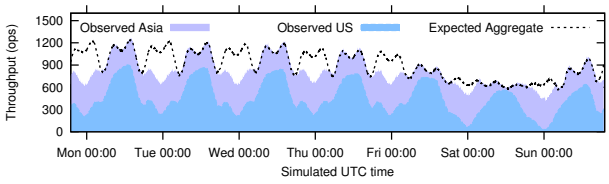
1. How effective is the scheduler in enabling applications to follow-the-sun?
2. Is VM migration a viable approach to follow-the-sun?
3. What is the efficacy of Supercloud storage and networking in supporting live VM migration?

5.1 Follow the Sun

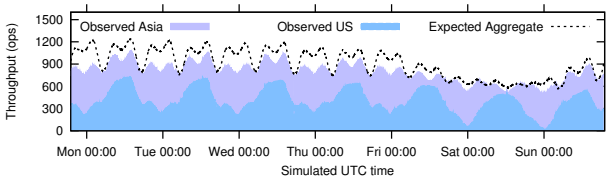
In this set of experiments, we use a distributed application, ZooKeeper, to investigate application performance benefits due to following the sun. Results demonstrate that the Supercloud scheduler was able to automatically follow the sun and migrate resources geographically and across heterogeneous clouds, enabling high performance to be maintained. For brevity, we have omitted experiments with a MySQL database, but we have similarly good results for those experiments as well.

ZooKeeper writes require a majority of ZooKeeper servers (aka, the ensemble) and a ZooKeeper leader to coordinate and order all writes. High network latency between ZooKeeper servers or between clients and the ZooKeeper leader causes high end-to-end service latencies. For good performance, the majority of ZooKeeper servers have to be located where most active clients are, and the ZooKeeper leader must be part of that majority.

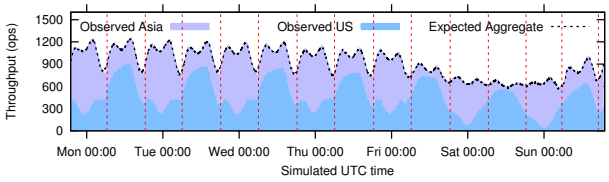
Experiment setup: To evaluate the efficacy of following the sun using the scheduler described in Section 3, we used a ZooKeeper distributed application and measured its ability to respond to clients in two different regions, Virginia and Taiwan. The clients used a workload corresponding to a weeklong trace of active MSN connections [21, Figure 5a]. The trace does not specify the start time of the MSN workload—we made an educated guess based on the diurnal pattern in the data and assumed that the workload started at 12am at the beginning of a Monday. Nor does the trace specify the absolute workload—we scaled the workload so that the peak workload can be served by our ZooKeeper cluster when latency is low. We varied load based on location and time: We circularly shifted the start of the trace by 12 hours ahead to produce an identical workload where the start of the trace from Virginia was 12 hours before starting in Taiwan. The ZooKeeper clients randomly connected to one ZooKeeper server and submitted blocking read and write requests in a ratio of 9:1. Each read operation obtained a 64-byte ZooKeeper *znode*; each write operation overwrote a



(a) US Ensemble

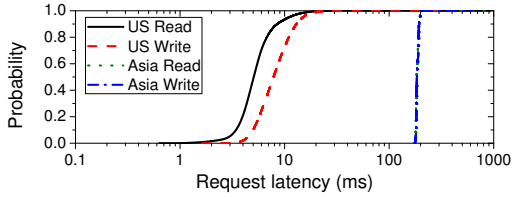


(b) Global Ensemble

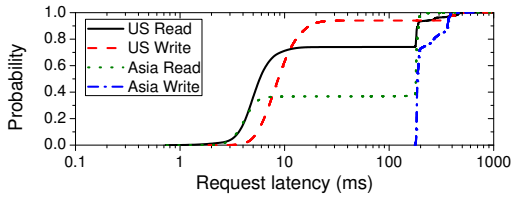


(c) Dynamic Ensemble

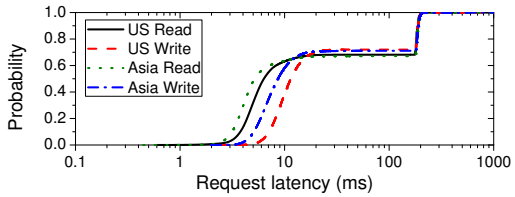
Figure 4. ZooKeeper throughput (vertical dashed lines indicate the end of the migrations).



(a) US Ensemble



(b) Global Ensemble



(c) Dynamic Ensemble

Figure 5. ZooKeeper latency CDF.

64-byte znode. The clients ran on first-layer VMs in Virginia on Amazon VMs and in Taiwan on Google VMs.

For the ZooKeeper ensemble, we deployed the servers in the Supercloud simultaneously spread across Amazon Virginia and Google Taiwan regions. The type of first layer VMs used in our experiments was `m3.xlarge` in Amazon and `n1-standard-4` in Google, both of which had 4 vCPUs and 15GB memory. The ZooKeeper ensemble ran on three second-layer VMs, denoted as `zk1`, `zk2` and `zk3`. Each VM had 1GB RAM and 1 CPU core. We used one first layer VM in each region to serve as the Supercloud storage server. The data of the ZooKeeper nodes was stored on disk and propagated automatically by the storage servers.

We implemented the scheduling evaluation functions for ZooKeeper discussed in Section 3. ZooKeeper server VMs reported VM load to the scheduler. The scheduler placement evaluation was triggered every minute. Once a VM migration was started, the scheduler waited until it finished before considering a new placement. Migrations were performed in parallel, so going from one placement plan to another was fast.

We evaluated ZooKeeper in three scenarios:

1. **US Ensemble**: all three ZooKeeper nodes, `zk1`, `zk2`, and `zk3`, were in Amazon Virginia;
2. **Global Ensemble**: `zk1` and `zk2` were in Amazon Virginia, and `zk3` was in Google Taiwan;
3. **Dynamic Ensemble**: the Supercloud scheduler automatically placed VMs according to the workload.

To simplify experimentation and save cost, we sped up the trace by a factor of 30 so that the one week trace could be replayed in less than six hours. The performance of the US Ensemble and Global Ensemble was not affected by the speed-up, while the performance of the Supercloud was slightly degraded since migration was not sped up correspondingly.

Experimental results: Figure 4 shows the throughput (in operations per second) for the three scenarios, US, Global, and Dynamic Ensembles. Each scenario displays throughput measured for the United States (US) clients in Virginia ("Observed US"), throughput measured for Asian clients in Taiwan ("Observed Asia"), and throughput if latency were negligible ("Expected Aggregate"). The throughput results observed by US and Asian clients are stacked on top of one another (with US below Asia), so that the top of the throughput results for the clients in Asia show the total aggregate throughput that was observed by all clients. For the US Ensemble scenario, Figure 4a, US clients were able to reach their maximum throughput, but clients in Asia suffered from high latencies, and, as a result, experienced poor throughput. For the Global Ensemble, Figure 4b, by placing one node in Taiwan, $1/3^{rd}$ of Asian clients experienced increased read throughput and total throughput was improved. However, $2/3^{rd}$ of the read requests and all write requests from Asian clients still experienced poor throughput. Moreover, US clients were not able to reach their maximum throughput during peak times because $1/3^{rd}$ of clients were connected to the ZooKeeper server in Taiwan.

Finally, Figure 4c, shows the result for the Supercloud case where throughput remained high and matched the expected performance as the scheduler was able to automatically migrate ZooKeeper servers to the region where load was high.

Figure 5 shows the cumulative latency distributions for read and write operations. We can see that for the US Ensemble, Figure 5a, 90% of US clients observed less than 15ms latency for read and write operations, while client latencies in Asia were close to 200ms. (We repeated the experiments with an Asia Ensemble and observed symmetric results.) In the Global Ensemble, Figure 5b, the ZooKeeper quorum was in the US. The server in Taiwan helped Asian clients gain better read throughput, though write latency did not improve: 37% reads from Asian clients completed in 15ms because they were handled by the server in Taiwan. The high tail latency of US read/write performance was because some requests went to the server in Taiwan. (A symmetric experiment with the quorum in Asia observed the same results.)

Finally, in the Dynamic Ensemble, Figure 5c, the scheduler automatically figured out that placing all three nodes in one location was the best placement for the workload. This is because clients connect to servers randomly and the read/write ratio is fixed. For example, leaving one node in Asia can only improve performance of one third of read requests from Asia, but also make one third of read requests from the US suffer from long latency. This was not beneficial when the US workload was higher than the Asia workload. Therefore, it always migrated the whole ensemble together: The scheduler migrated the ensemble every 12 hours between Amazon Virginia and Google Taiwan to make the ZooKeeper cluster close to most clients. The scheduler placement resulted in low latency: 61% of writes and 69% of reads were less than 15ms. The tail latency was caused by some requests going to a remote server.

5.2 Comparing Migration Approaches

In this second set of experiments, we use two popular distributed applications, Cassandra and ZooKeeper, to investigate the viability of relying on live VM migration via the Supercloud to enable distributed applications to follow-the-sun. The results demonstrate that not only can distributed applications benefit from the Supercloud, but they can also do so without any change to the application and can outperform other approaches that require application modification.

5.2.1 Cassandra Migration

Cassandra [20] supports adding and removing nodes and automatically handles data replication and load distribution. When following the sun, it makes little sense to move only some of the Cassandra nodes. Migrating an entire Cassandra cluster can be implemented by adding nodes in the destination location and removing nodes in the source location.

Experiment setup: To compare migration approaches, "application" and "Supercloud", we started a 3-node Cassandra cluster in the Amazon Virginia region, then migrated the whole cluster to the Google Taiwan region. As a result, the migration was across geographically separated clouds. We deployed a 3-node Cassandra cluster with replication factor of 2 in second-layer VMs with 1vCPU and 2GB memory. For the first-layer VMs, we used `m3.xlarge` instances in the Amazon Virginia region and `n1-standard-4` instances in the Google Taiwan region.

For the application migration approach, a Cassandra cluster was initially running in Virginia. To move all nodes to Taiwan, we followed the process specified in the Cassandra documentation for replacing running nodes in the cluster. Three new nodes in Taiwan joined the cluster with a configuration file pointing to a seed node and automatically propagated their information through Cassandra's gossip protocol. The key space then spread evenly across all six nodes and the data associated with the key moved automatically. After the three new nodes joined the cluster, we performed "node decommissioning" in each of the three original nodes. As a result, the Virginia nodes copied their data to the Taiwan nodes. The gossip protocol automatically updated each node with the cluster information transparently to the clients.

For the Supercloud migration, we set up a Supercloud across the Amazon Virginia and Google Taiwan regions using the same first- and second-layer VM configurations described above.

For the workload, data was stored in memory and moved explicitly with the application approach or migrated with the VM with the Supercloud approach. We populated the database with 30,000 key/value pairs, each of which had a size of 1KB. We started one client in each region. The clients read and wrote to the database continuously with a ratio of 4:1; each read operation obtained the value of a random key, and each write request updated the value of a random key. The consistency level was set to `ONE` (default), indicating eventual consistency.

Experimental results: Figure 6(a) shows the throughput of application migration (in operations per second) for both Amazon Virginia and Google Taiwan clients. Although not easy to see in this case, the throughput results for both types of clients are again stacked on top of one another, with the throughput of the Asia clients below that of the throughput of the US clients. The top of the graph therefore shows the total aggregate throughput. During migration, the throughput dropped dramatically and remained low for about 200 seconds. Even after the migration completed, it took another 200 seconds to restore performance to the original throughput due to the overhead of data replication.

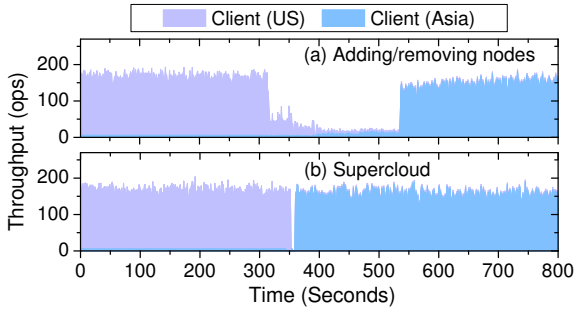


Figure 6. Comparison of different migration mechanisms for moving a Cassandra cluster.

With the VM migration mechanism in the Supercloud, we migrated the three Cassandra VMs from Amazon Virginia to Google Taiwan in parallel. Figure 6(b) shows that performance impact was small with a downtime of around 5 seconds. (Downtime could be reduced further by synchronizing the migration finishing time—a project we plan for future work.) The Supercloud maintained the same IP addresses and network topology, without triggering any unnecessary data replication or key shuffling.

5.2.2 ZooKeeper Migration

Experiment setup: To evaluate different follow-the-sun approaches for ZooKeeper, we set up a Supercloud using Amazon EC2 `m3.xlarge` instances (4 vCPUs, 15GB memory) in the Virginia and Tokyo regions. We deployed a ZooKeeper ensemble in three second-layer VMs with 1 vCPU and 1GB memory. The leader was initially in Virginia, with one follower in Virginia and another in Tokyo. We started one client in each region generating a constant workload with read/write ratio 9:1. Each read operation obtained a 1KB ZooKeeper znode; each write operation overwrote a 1KB znode.

We compared three approaches.

1. A "2-step reconfiguration" moved a majority of the servers from one region to another: We first added a new node in Tokyo, then removed the original leader in Virginia.

   Unfortunately, the 2-step reconfiguration does not guarantee that a node in Tokyo will be elected as the leader. Instead, it was more likely that one of the Virginia nodes would become the leader, which was not desirable since the majority of the ensemble is in Tokyo.

2. A "3-step reconfiguration" ensured that the leader ended up in Tokyo while maintaining the same level of fault tolerance: We added two nodes in Tokyo first, then removed both nodes from Virginia. After the new leader was elected in Tokyo, we added a new node in Virginia and removed one of the nodes from Tokyo.

3. Using the Supercloud we transparently migrated a second-layer VM running the leader from Virginia to Tokyo. Neither ZooKeeper nor clients required any modification.

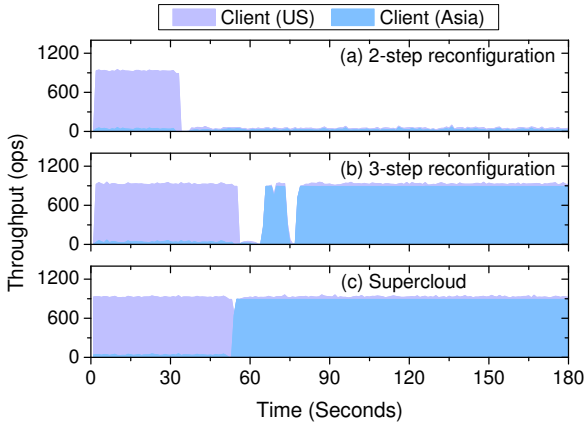


Figure 7. Comparison of different migration mechanisms for moving the ZooKeeper leader.

Experimental results: Figure 7 shows the stacked throughput (in operations per second) of clients in both Virginia and Tokyo regions before and after leader migration using the three migration approaches discussed above. After the 2-step reconfiguration shown in Figure 7(a), the leader role was switched to a Virginia node while the two followers were in Tokyo, which was inefficient and, as a result, the throughput dropped significantly for both clients. Using the 3-step reconfiguration in Figure 7(b), the leader was successfully moved to Tokyo so good throughput for the Tokyo clients were achieved. Unfortunately, the 3-step reconfiguration took 20 seconds during which performance was inconsistent and low. Finally, Figure 7(c) shows the performance of the Supercloud: the drop in performance was less than a second and transparent to the ZooKeeper application and its clients.

5.3 Storage Evaluation

Experiment setup: To evaluate storage performance under migration, we deployed a Supercloud setup with two `m3.xlarge` instances in Amazon Virginia and Northern California regions. The ping latency between two VMs in these regions was 75ms. We started a user VM with 1 virtual CPU core and 512MB memory and ran the `DBENCH` [23] benchmark in it, which simulated four clients generating 500,000 operations each on the filesystem based on the standard `NetBench` benchmark. We configured the DBENCH clients so that they ran at the fastest possible speed. Without migration, a VM using local storage finished the benchmark in two minutes.

In the following experiments, immediately after starting the benchmark, we triggered a VM migration from Virginia to Northern California. The full migration took 40 to 50 seconds (most of which was in the background while the VM kept running). During most of the full migration, in

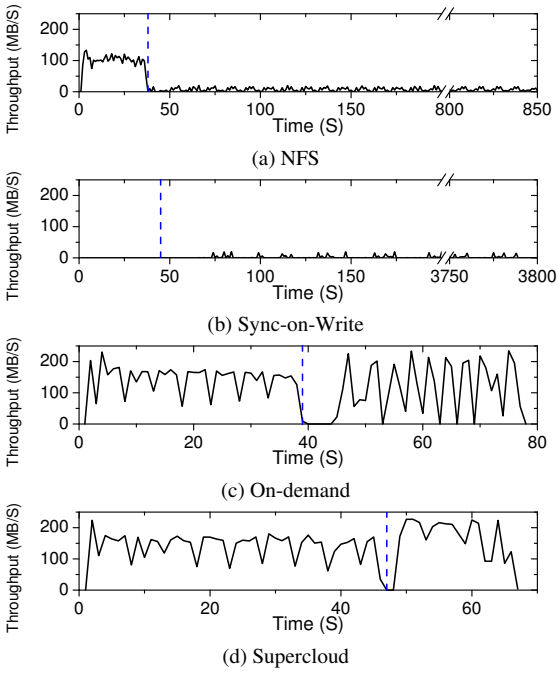


Figure 8. Average throughput per second of the DBENCH benchmark in the migrated VM. For clarity, the x-axes are on different scales.

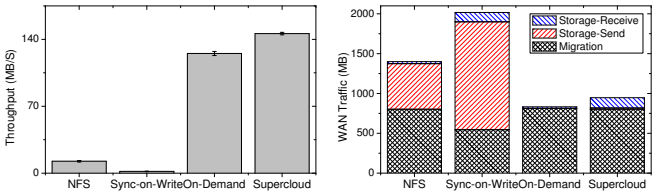
particular during the pre-copy phase, the benchmark kept running at the old location. In fact, before the VM was actually moved to Northern California, one third of the workload had finished.

We compared the performance of the benchmark with four different underlying storage systems:

- NFS: A traditional NFS server deployed in Amazon Virginia.
- Sync-on-Write: A strongly consistent geo-replicated store that synchronously propagates each write.
- On-demand: Supercloud storage with proactive propagation turned off.
- Supercloud: Supercloud storage with proactive propagation in the background.

Except for NFS, all schemes were implemented in the Supercloud's propagation manager for fair comparison.

Experiment result: Figure 8 shows the results: The average throughput in each second of the DBENCH benchmark. The vertically dashed lines indicate when migration was finished. Figure 8a, NFS, shows that throughput dropped significantly after the clients were migrated. Low performance resulted from remote disk accesses that incurred high latency. Figure 8b, sync-on-write, shows that a strongly consistent geo-replicated storage incurs high performance overhead both before and after migration because each write needs to be propagated through the wide area network. Figure 8c, Supercloud



(a) DBENCH throughput (b) WAN traffic on the migration destination

Figure 9. Average throughput and total WAN traffic of the DBENCH benchmark in the migrated VM.

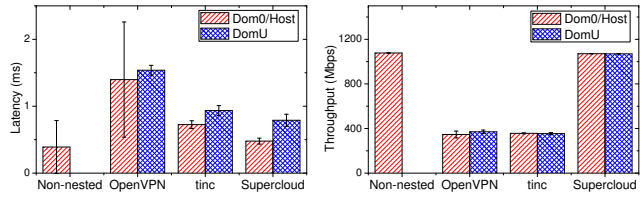
on-demand, eventually achieved good average throughput after migration since all writes could be committed locally. However, read throughput was low right after migration since no blocks had been copied ahead of time. Finally, Figure 8d, Supercloud proactive propagation, shows that most read and write requests could be served locally. Consequently, the corresponding benchmark took the least time.

Figure 9a shows the average throughput for each case. We repeated the same experiment three times and report the average number with standard deviation. The Supercloud proactive propagation achieved the highest throughput.

Figure 9b shows the total WAN traffic measured at the migration destination (Northern California). The migration caused 850MB of WAN traffic for NFS, On-demand, and Supercloud, but only 570MB of traffic for Sync-on-Write. In particular, the migration traffic depended on the page dirty rate of the VM. A benchmark running at a higher speed caused more dirty memory pages to be copied during migration. Sync-on-Write storage results in low throughput in the benchmark, thus lowering the page dirty rate of the whole VM. Both NFS and Sync-on-Write incur a lot of outgoing traffic when accessing storage because each disk write needed to go through the WAN. Sync-on-Write also had higher incoming traffic, which was generated when the benchmark was running in Virginia. On-demand achieved the lowest WAN traffic since it fetched data remotely only when necessary, and no update propagation was needed. The Supercloud incurred more incoming traffic than On-demand because it pushed some data that was not needed, but it achieved higher throughput and predictable performance.

5.4 Network Evaluation

To evaluate the network performance of the Supercloud, we deployed the Supercloud in two `m3.xlarge` instances in the Amazon Virginia region and measured UDP latency (using UDP ping) and TCP throughput (using `netperf`) between second-layer Domain-0 VMs and Domain-U VMs respectively. For comparison, we ran the same benchmark using the following settings:



(a) UDP latency (b) TCP throughput

Figure 10. Network performance evaluations.

- Non-nested: a setup where we ran the benchmark directly in the first-layer VMs for baseline purposes.
- OpenVPN: a VPN solution using a centralized controller, also running in Amazon Virginia.
- tinc: a P2P VPN solution, which implements full-mesh routing.
- Supercloud: the Supercloud implementation based on Open vSwitch.

For the latency test, we ran 50 UDP pings (1 ping per second). Figure 10a shows the average latency and standard deviation across all runs. Although our Open vSwitch-based virtual network slightly increases the UDP latency, the overhead was much smaller than either OpenVPN or tinc. The latency for dom0 was smaller than domU because network packets to DomU go through Dom0 first. OpenVPN had the highest latency because all packets travel through the centralized controller.

To measure TCP throughput, we enabled jumbo frames in all setups and repeated a `netperf` TCP stream benchmark with default options 10 times for each setup. As shown in Figure 10b, the throughput of non-nested instances, Dom0, and DomU in the Supercloud all achieved 1Gbps with small variance. In contrast, tinc and openVPN could only achieve 300Mbps. Because both tinc and OpenVPN use a tap device connected to a user-level process, the extra memory copy and kernel-user mode context switching incurred significant overhead.

6. Related Work

Multi-cloud deployment is attractive to users because of its high availability and cost-effectiveness. SafeStore [35], DepSky [13], HAIL [16], RACS [10], SPANStore [50], Hybris [24] and SCFS [14] have demonstrated benefits of multi-cloud storage, but they do not support computational resources. Docker [2] deploys applications encapsulated in Linux containers (LXC) to multiple clouds. Although light-weight, LXC is not as flexible as a VM because all containers must share the same kernel, and it has poor migration support. Ravello [7] leverages a nested hypervisor to provide an encapsulated environment for debugging and development distributed applications, but it does not addresses the challenges of providing storage and network support for wide-area application migration. Platforms like fos [46], Rightscale [8], AppScale [1], TCloud [45], IBM Altocumulus [37], and Conductor [47] enable multi-cloud application deployment, but none of them provides the generality and flexibility of a multi-cloud IaaS.

VM live migration [22, 29] has been widely used for resource consolidation and workload burst handling [18, 28, 39]. However, VM live migration is not exposed to end users of public clouds. Migrating a VM in the wide-area network faces long latency in accessing a shared disk image. Previous studies [11, 15, 17, 27, 30, 36, 40, 49, 52] proposed various solutions for optimizing the migration of the whole VM disk image, while our Supercloud image storage attempts to only transfer data that is necessary. FVD [43] is a new VM image format that supports copy-on-write, copy-on-read, and adaptive prefetching that can be used for optimizing the performance of migrating a VM with the disk image. However, FVD can prefetch data only after the VM resumes on the destination. In contrast, the Supercloud image storage proactively propagates data before migration is triggered.

7. Conclusion

In this paper, we have demonstrated that it is important for global cloud services to be able to "follow the sun," dealing with diurnal workloads by migrating key servers to remain close to active users. We show that existing reconfiguration options in distributed services are not well-suited to support such migration, resulting in high overheads and long performance hiccups.

The Supercloud presents a complete cloud software stack under the user's full control that can seamlessly span multiple availability zones and cloud providers, including private clouds. It features live migration, as well as shared storage, virtual networking, and automated scheduling of workloads, placing and migrating VM resources as needed. Spanning availability zones and cloud providers, the Supercloud provides maximal flexibility for placement. Using our automated schedulers, we demonstrate continuous low latency for diurnal workloads.

Availability

The code of the Supercloud project is publicly available at `http://supercloud.cs.cornell.edu`.

Acknowledgments

This work was partially funded and supported by a SLOAN Research Fellowship received by Hakim Weatherspoon, DARPA MRC (FA8750-10-2-0238, FA8750-11-2-0256), DARPA CSSG (D11AP00266), NSF CSR-1422544, NSF CNS-1601879, NSF CAREER (1053757), NSF TRUST (0424422), NIST Information Technology Laboratory/Advanced Network Technologies Division (60NANB15D327), Cisco, Intel, Facebook, and Infosys. We would like to thank the anonymous reviewers for their comments.

References

[1] AppScale: The Open Source App Engine. http://www.appscale.com/.

[2] Docker. https://www.docker.com/.

[3] Marissa Mayer at Web 2.0. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html.

[4] Open vSwitch. http://openvswitch.org/.

[5] OpenStack. http://www.openstack.org/.

[6] OpenVPN. https://openvpn.net/.

[7] Ravello Systems. http://www.ravellosystems.com/.

[8] Rightscale. http://www.rightscale.com.

[9] The Cost of Latency. http://perspectives.mvdirona.com/2009/10/the-cost-of-latency/.

[10] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A case for cloud storage diversity. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*, pages 229–240, New York, NY, USA, 2010. ACM.

[11] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu. VMFlock: Virtual Machine Co-migration for the Cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 159–170, New York, NY, USA, 2011. ACM.

[12] NSF Award #1541215: Data Analysis and Management Building Blocks for Multi-Campus Cyberinfrastructure through Cloud Federation. https://www.nsf.gov/awardsearch/showAward?AWD_ID=1541215&HistoricalAwards=false.

[13] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and Secure Storage in a Cloud-of-clouds. In *Proceedings of the Sixth European Conference on Computer Systems (EuroSys'11)*, pages 31–46, New York, NY, USA, 2011. ACM.

[14] A. Bessani, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, and P. Verissimo. SCFS: A Shared Cloud-backed File System. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 169–180, Philadelphia, PA, June 2014. USENIX Association.

[15] S. K. Bose, S. Brock, R. Skeoch, and S. Rao. CloudSpider: Combining Replication with Scheduling for Optimizing Live Migration of Virtual Machines Across Wide Area Networks. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 13–22, Washington, DC, USA, 2011. IEEE Computer Society.

[16] K. D. Bowers, A. Juels, and A. Oprea. HAIL: A High-availability and Integrity Layer for Cloud Storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*, pages 187–198, New York, NY, USA, 2009. ACM.

[17] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live Wide-area Migration of Virtual Machines Including Local Persistent State. In *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07)*, pages 169–179, New York, NY, USA, 2007. ACM.

[18] R. Bryant, A. Tumanov, O. Irzak, A. Scannell, K. Joshi, M. Hiltunen, A. Lagar-Cavilla, and E. de Lara. Kaleidoscope: Cloud Micro-elasticity via VM State Coloring. In *Proceedings of the Sixth European Conference on Computer Systems (EuroSys'11)*, pages 273–286, New York, NY, USA, 2011. ACM.

[19] R. Buyya, R. Ranjan, and R. N. Calheiros. *Algorithms and Architectures for Parallel Processing: 10th International Conference, ICA3PP 2010, Busan, Korea, May 21-23, 2010. Proceedings. Part I*, chapter InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, pages 13–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[20] Cassandra Documentation: Replacing a Dead Node. https://docs.datastax.com/en/cassandra/2.0/cassandra/operations/ops_replace_node_t.html.

[21] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In *NSDI*, volume 8, pages 337–350, 2008.

[22] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation (NSDI'05)*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[23] DBENCH benchmark. https://dbench.samba.org/.

[24] D. Dobre, P. Viotti, and M. Vukolić. Hybris: Robust Hybrid Cloud Storage. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'14)*, pages 12:1–12:14, New York, NY, USA, 2014. ACM.

[25] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker. Frenetic: A high-level language for openflow networks. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, pages 6:1–6:6, New York, NY, USA, 2010. ACM.

[26] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization (IISWC '07)*, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.

[27] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. Shenoy. Seagull: Intelligent Cloud Bursting for Enterprise Applications. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 33–33, Berkeley, CA, USA, 2012. USENIX Association.

[28] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: A Consolidation Manager for Clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'09)*, pages 41–50, New York, NY, USA, 2009. ACM.

[29] M. R. Hines and K. Gopalan. Post-copy Based Live Virtual Machine Migration Using Adaptive Pre-paging and Dynamic Self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 51–60, New York, NY, USA, 2009. ACM.

[30] T. Hirofuchi, H. Nakada, H. Ogawa, S. Itoh, and S. Sekiguchi. A Live Storage Migration Mechanism over Wan and Its Performance Evaluation. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '09, pages 67–74, New York, NY, USA, 2009. ACM.

[31] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Annual Technical Conference*, volume 8, page 9, 2010.

[32] Q. Jia, Z. Shen, W. Song, R. van Renesse, and H. Weatherspoon. Supercloud: Opportunities and challenges. *SIGOPS Oper. Syst. Rev.*, 49(1):137–141, Jan. 2015.

[33] Q. Jia, Z. Shen, W. Song, R. van Renesse, and H. Weatherspoon. Smart Spot Instances for the Supercloud. In *CrossClouds*, April 2016.

[34] F. P. Junqueira, B. C. Reed, and M. Serafini. Zab: High-performance broadcast for primary-backup systems. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 245–256. IEEE, 2011.

[35] R. Kotla, L. Alvisi, and M. Dahlin. SafeStore: A Durable and Practical Storage System. In *Proceedings of the USENIX Annual Technical Conference (ATC'07)*, pages 10:1–10:14, Berkeley, CA, USA, 2007. USENIX Association.

[36] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai. The Design and Evolution of Live Storage Migration in VMware ESX. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.

[37] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. Anderson. IBM Altocumulus: a cross-cloud middleware and platform. In *Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA'09)*, pages 805–806, New York, NY, USA, 2009. ACM.

[38] A. Mirkin, A. Kuznetsov, and K. Kolyshkin. Containers checkpointing and live migration. In *Proceedings of the Linux Symposium*, volume 2, pages 85–90, 2008.

[39] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 69–82, San Jose, CA, 2013. USENIX.

[40] B. Nicolae and F. Cappello. A Hybrid Local Storage Transfer Scheme for Live Migration of I/O Intensive Workloads. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 85–96, New York, NY, USA, 2012. ACM.

[41] A. Shraer, B. Reed, D. Malkhi, and F. Junqueira. Dynamic reconfiguration of primary/backup clusters. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, Berkeley, CA, USA, 2012. USENIX Association.

[42] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 275–287, New York, NY, USA, 2007. ACM.

[43] C. Tang. FVD: A High-performance Virtual Machine Image Format for Cloud. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2011. USENIX Association.

[44] tinc VPN. `http://www.tinc-vpn.org/`.

[45] P. Verissimo, A. Bessani, and M. Pasin. The TClouds architecture: Open and resilient cloud-of-clouds computing. In *Dependable Systems and Networks Workshops (DSN-W), IEEE/IFIP 42nd International Conference on*, June 2012.

[46] D. Wentzlaff, C. Gruenwald, III, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal. An operating system for multicore and clouds: mechanisms and implementation. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*, pages 3–14, New York, NY, USA, 2010. ACM.

[47] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the Deployment of Computations in the Cloud with Conductor. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*, pages 27–27, Berkeley, CA, USA, 2012. USENIX Association.

[48] D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize once, run everywhere. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*, pages 113–126, 2012.

[49] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe. CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'11)*, pages 121–132, New York, NY, USA, 2011. ACM.

[50] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. SPANStore: Cost-effective Geo-replicated Storage Spanning Multiple Cloud Services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*, pages 292–308, New York, NY, USA, 2013. ACM.

[51] G. Zhang, L. Chiu, and L. Liu. Adaptive data migration in multi-tiered storage based cloud environment. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 148–155, July 2010.

[52] J. Zheng, T. S. E. Ng, and K. Sripanidkulchai. Workload-aware Live Storage Migration for Clouds. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, pages 133–144, New York, NY, USA, 2011. ACM.