Inception: Towards a Nested Cloud Architecture

Changbin Liu   Yun Mao
AT&T Labs - Research

Abstract

Despite the increasing popularity of Infrastructure-as-a-service (IaaS) clouds, providers have been very slow in adopting a large number of innovative technologies, such as live VM migration, dynamic resource management, and VM replication. In this paper, we argue that the reasons are not only technical but also fundamental, due to lack of transparency and conflict of interest between providers and customers. We present our vision *inception*, a nested IaaS cloud architecture to overcome this impasse. Inception clouds are built entirely on top of the resources acquired from today's clouds, and provide nested VMs to end users. We discuss the benefits, use cases, and challenges of inception clouds, and present our network design and prototype implementation.

1 Introduction

Infrastructure-as-a-service (IaaS) clouds provide on-demand programmable access to a large pool of elastic compute, network and storage resources in the form of virtual machines (VMs). The increasing popularity of cloud also becomes in part the driving force of a large number of innovative technologies at the hypervisor level in the areas of dynamic resource allocation [8, 9, 11, 16], high availability [14], security [10], etc. Unfortunately, we have found few of these technologies deployed in the public clouds. For example, live VM migration [9], a technology developed for years and shipped in both stock Xen and KVM stable branches, is not supported in most public clouds. When a physical server is scheduled for maintenance, users are notified to manually shutdown their VMs on the server and boot up elsewhere.

We argue that such obstacles against rapid innovation do not exist by coincidence. While sometimes a new technology is not commercially adopted due to its experimental nature, it is also fundamentally challenging, or virtually impossible in many cases, due to lack of transparency and conflict of interest between providers and customers. For example, live VM migration typically requires the source and destination physical servers to be on the same layer-2 network, and have shared storage for VM images. While the feature is desirable for some applications, it is not required for others (e.g., horizontally scalable stateless web servers). Since it is often easier and cheaper to build a scalable layer-3 network without shared storage, providers have to weigh the tradeoff between features that are only useful for some applications, and the tax they impose on the infrastructure for all applications.

On the other hand, instead of using a general-purpose public cloud, one could also install cloud management software (CMS) such as OpenStack [4] and vCloud [5] to build a private cloud, and customize the software stack to her specific needs. However, only mega-size companies have the capability to achieve the scale and multiple physical points of presence of the public clouds. The upfront investment to build a private cloud including hardware, software and operation cost might also be impractical for many organizations.

In this paper we present *inception*, a nested IaaS cloud architecture to overcome this impasse. Unlike today's clouds (referred to as "reality clouds" for the rest of the paper) that built with physical resources, inception clouds are built entirely on top of the resources encapsulated in VMs acquired from reality clouds, without any provider cooperation or involvement. Compared to reality clouds, inception clouds provide exactly the same abstractions (e.g., VMs, block storage, networks) to customers. Inception clouds not only inherit the benefits of elasticity and geographic diversity from reality clouds, but also can be tailored according to specific user scenarios and workloads. Our grand vision is that inception clouds will significantly lower the barrier to build, instrument, evaluate, and deploy cloud technologies, therefore accelerate the innovation in the area.

Inception is based on nested virtualization [7, 18, 19], a topic that receives increasing attention in the research community. Similar to inception, xCloud [17] also considered nested virtualization as one of its design choices to build nested clouds to achieve hypervisor extensibility. The contribution of this paper is to examine the full stack vertically and horizontally beyond a single host, including not only server virtualization, but also networking design and CMS, identify use cases and challenges, present a viable solution based on software-defined networking (SDN) and OpenStack, and layout future roadmap.

2 Overview

An inception cloud provider hosts the entire infrastructure on top of VMs acquired from reality clouds (rVMs), and produce inception VMs (iVMs) that are nested inside the rVMs. While we expect there are only a few large companies as reality cloud providers with huge data centers and geographic diversity just like today's public cloud space, we envision that inception cloud providers are much more diverse in size and specialty, and each inception cloud is specially optimized for a few types of applications. Interestingly, an organization might also be both inception cloud provider and customer given enough technical depth to enjoy a fast feedback loop for optimization and customization.

2.1 Benefits

There are several fundamental benefits of inception clouds compared with reality clouds:

- Low operation overhead. Because there is no physical hardware to manage, inception cloud providers do not need to worry about issues like cooling, power, cabling, rack design, malfunctioned hardware replacement, data center real estate, etc. This significantly lowers the barrier to run a cloud.

- Dom0/hypervisor flexibility. Inception cloud providers have the flexibility to choose hypervisor features such as live VM migration [9] and replication [14], or value added services like intrusion detection [10] and customized firewall at Dom0 without any cooperation of reality cloud providers.

- App-specific resource allocation. App-specific inception clouds can take workload characteristic into account to optimize resource allocation. For example, a real-time application is very sensitive to oversubscribed resources. On the other hand, a performance insensitive application staying idle most of the time might enjoy high subscription ratio to improve overall resource utilization. A general-purpose reality cloud treats VMs as black boxes. So it is very complicated if not impossible to distinguish those applications and optimize accordingly, not to mention multiple potentially conflicting service level agreements.

- Multiple providers. An inception cloud may span across multiple reality clouds, such as a private on-premise one and a public one, or among different public providers. This opens up an optimization opportunity to take advantage of geographic approximation or cost reduction.

2.2 Use Cases

We illustrate three representative use cases of inception clouds as shown in Figure 1. They are built on top of two public and one private reality clouds, distributed in East, West and South Coast of the US, respectively.

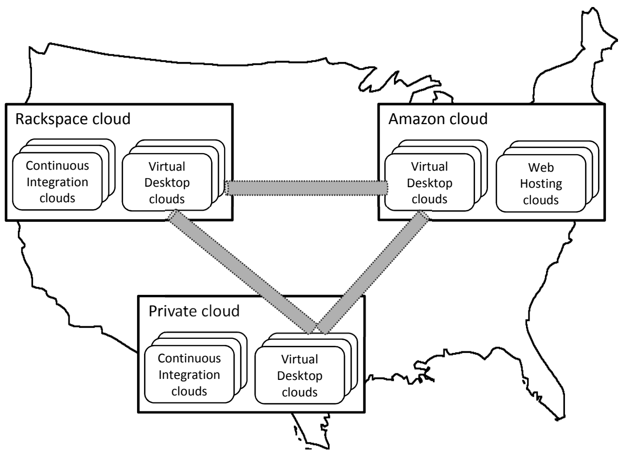

Figure 1: Inception cloud use cases.

Virtual Desktop. In a virtual desktop inception cloud, each iVM represents an employee's desktop. The interactive nature demands low latency from the client device to its iVM. The desktop iVMs can also be consolidated efficiently because they share many similar memory pages and disk images, and require low CPU utilization. To build a virtual desktop inception cloud, we choose all three reality clouds as points of presence. The iVM allocated to an employee will be transparently live migrated to the closest physical location when she travels. At the hypervisor level, we enable memory reduction features [11, 16] and set CPU over-subscription ratio to high. We also run a virus scanner at Dom0 to periodically scan iVM disks and quarantine infected iVMs if any.

Web Hosting. In a web hosting cloud, we expect to host two major types of iVMs: web servers and database servers. Unlike virtual desktops, both servers are performance critical and do not work well with high resource subscription ratio. Moreover, many web servers are stateless, and can scale up and down by adding or removing instances. So they do not need live migration. We design the web hosting cloud by enabling two hypervisor features: instant VM spawning [8] for web servers to deal with flash crowd, and VM replication [14] to achieve transparent database high availability. Optionally, one can also run an intrusion detection system [10] on Dom0 for all public facing web server iVMs.

Continuous Integration. In the development cycle of CMS, continuous integration (CI) executes a series of integration tests for every patch submitted and rejects it unless all tests pass. Take OpenStack for example, due to the complexity to setup and tear down test environments, all integration tests are executed in VMs. Currently, all tests are executed within a single KVM-based VM. The VMs spawned during testing are nested inside KVM based on QEMU. Unfortunately, this setup is far from ideal to test all complicated setups in networking and scheduling. Instead, in our CI inception cloud, each OpenStack code snapshot is the CMS, and a CI cloud instance only exists shortly during the testing phase. Our flexible networking design (Section 3) allows running all complicated multi-node setup for current-version OpenStack. The official OpenStack CI environment and any enterprise who customizes OpenStack code could use CI inception clouds for comprehensive testing.

2.3 Challenges

Realizing inception clouds faces many challenges. First, although recent work [7, 18] brought the nested virtualization overhead in many cases within 5% to 10% of paravirtualized drivers in a single-level paravirtualized guest, there are still room for improvement, especially in I/O performance.

Furthermore, hypervisor is only one piece of the software puzzle to operate a cloud. A full stack of solutions including cloud controller, configuration, monitoring and troubleshooting are needed. Recent advances in the open source world such as OpenStack lower the barrier but there are still many open questions. For example, inception clouds run on rVMs instead of physical servers. The membership of a rVM cluster could be a lot more dynamic than a physical cluster. OpenStack is not designed for it, and of course not optimized for it.

Last but not the least, the networking setup from today's public reality clouds differs dramatically. Each rVM might have one or more network interfaces plugged into layer-2 or layer-3 domains, various degrees of connectivity to private networks or the Internet, and different firewall rules. To make an inception cloud independent of a particular reality cloud networking solution, we assume the following "least common denominator": each rVM has one interface with a pre-assigned private IP address, and an optional floating public IP address, reachable directly via network address translation (NAT). Only TCP and UDP unicast traffic is allowed and no IP spoofing. Such setup is available in most if not all of today's public clouds [1, 4]. However, it is very different from how physical servers are wired in data centers, and is incompatible with many CMS such as OpenStack. Besides, setting up the networks for rVMs from different providers to run in a same inception cloud is a challenge.

3 Network Design

In this section we focus on the network design of an inception cloud to run stock CMS like OpenStack.

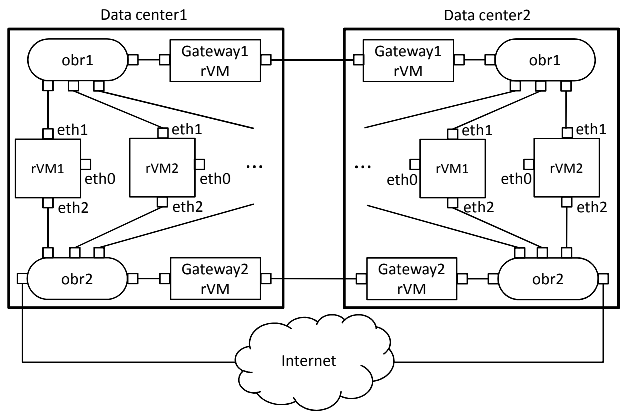

Figure 2: Network design of an inception cloud.

Figure 2 gives an overview of the network design of an inception cloud from the perspective of two reality cloud data centers. The data centers are geographically distributed. In an inception cloud, each rVM is equipped with three sets of isolated networks–management, private and public, marked as `eth0`, `eth1` and `eth2` respectively in the figure. The management network `eth0` corresponds to an intra-data center layer-2 or layer-3 network provided by reality clouds for rVM-to-rVM communications. Moreover, we adopt Open vSwitch [15] and tunneling protocols (e.g., VXLAN [6], GRE [3]) to build virtual switches and construct a layer-2 network for private iVM-to-iVM communications over `eth1`, as well as a layer-2 network for public access to iVMs from the Internet over `eth2`. In Figure 2 the `eth1` and `eth2` interfaces of each rVM are connected to virtual switches `obr1` and `obr2`, respectively. In addition, in each data center two gateway rVMs with public IPs provide inter-data center connections for `eth0` and `eth1` networks.

We note that the three networks–management, private, and public, are isolated from each other. Their mechanism is transparent to end users of an inception cloud. As a result, security issues like eavesdropping are eliminated by design. This network design further enables an inception cloud to construct layer-2 private/public networks with arbitrary topology for whatever reasons. Open vSwitch also supports the OpenFlow protocol such that more scalable layer-2 technologies can be easily adopted [12, 13]. One can also run multiple gateway rVMs as ingress/egress routers to avoid single point failures.

3.1 Private Network

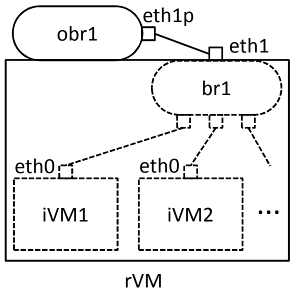

Figure 3: Private network of an inception cloud. Components generated by CMS are marked with dashed lines.

Figure 3 shows how iVMs are attached to the private network `eth1` from the perspective of a single rVM. `obr1` is the local vSwitch running on each rVM. Moreover, on each rVM we create a veth pair `<eth1, eth1p>`, with one end attached to `obr1`, and the other end `eth1` available for CMS network controller to build its private network in the inception cloud, as if it were an actual physical interface. Components that are generated by CMS are marked with dash lines in the figure. For example, if one chooses OpenStack Nova network with the flat DHCP mode, a Linux bridge `br1` is auto-created. `eth1` and all interfaces of each iVM are attached to `br1`. The IP addresses of iVMs are assigned by a DHCP server.

We specifically rejects the design where an iVM interface is directly plugged into `obr1` because (1) it would force CMS network controller to work with Open vSwitch, which could be incompatible with many existing networking solutions based on Linux bridge; (2) a network controller usually creates and manages virtual switches on its own so it does not work with `obr1` created out of band; (3) Open vSwitch does not work properly with ebtables and iptables in certain scenarios, which are used widely to implement firewall policies.

3.2 Public Network

Providing Internet access to iVMs is challenging. In today's reality clouds [1, 4], Internet access to rVMs is often achieved by setting up one or multiple network controllers, which perform NAT between public IPs and rVMs' private IPs. This appears that "floating" public IPs can be dynamically associated with specified rVMs.

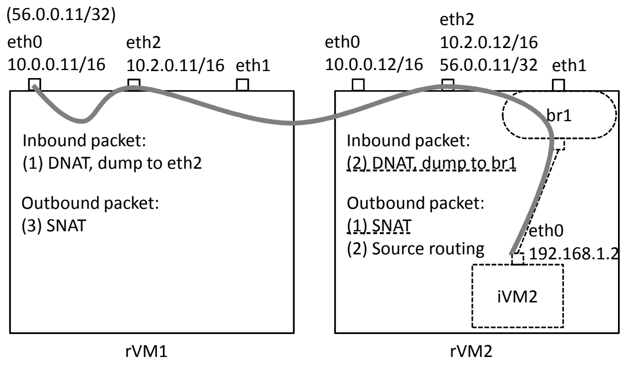

Figure 4: Public network of an inception cloud. Components generated by CMS are marked with dashed lines. The gray line indicates the traffic path between a iVM and the Internet.

To provide the same feature of floating IP association in an inception cloud, we design a public network setup, which requires no changes to underlying reality clouds, is transparent to end-users, and even works across data centers. We illustrate this design via an example given in Figure 4. There are two reality VMs–rVM1 and rVM2. As noted before, `eth0` is on the management network, e.g., with subnet 10.0.0.0/16. `eth2` is plugged in a virtual layer-2 network for public traffic with subnet 10.2.0.0/16. Suppose rVM1 has an associated floating public IP 56.0.0.11 assigned from its reality cloud, and we would like to associate this public IP to iVM2 on rVM2.

Floating IP on rVM: First, we would like to give the inception cloud the ability to simply configure a public IP address on `eth2` on the rVM to send and receive traffic with that IP. In the example, 56.0.0.11/32 is associated on `eth2` at rVM2. For an inbound packet sourced from the Internet and destined to 56.0.0.11, first rVM1 receives it on `eth0` after the packet being NATed by reality cloud network controllers, i.e., the original destination 56.0.0.11 is replaced with 10.0.0.11. Second, rVM1 performs a destination NAT (DNAT) to reverse this effect, i.e., destination 10.0.0.11 is substituted to 56.0.0.11, and the packet is dumped to `eth2` and forwarded to the public network via `obr2`. Finally the packet reaches `eth2` on iVM2 via `obr2`. On the outbound direction, we setup a source routing rule on rVM2, such that all traffic with source IP 56.0.0.11 should be routed via next hop 10.2.0.11. As a result, the packet will go through `obr2` and reach rVM1 on `eth2`. On rVM1, there is a source NAT (SNAT) rule to replace the source IP 56.0.0.11 with 10.0.0.11 and send out via `eth0` to reach the Internet via the reality cloud network.

Floating IP on iVM: After a rVM can freely configure public IP address, it becomes trivial to use a cloud network controller without modification to implement floating IPs for iVMs. We use OpenStack Nova network as an example to show the complete picture in Figure 4. Same as in the private network case, components generated by CMS are marked with dashed lines. Linux bridge `br1` is created and `eth0` on iVM2 is plugged in it, with an assigned IP 192.168.1.2. For an inbound Internet packet, once rVM2 receives it, a DNAT rule will translate 56.0.0.11 to 192.168.1.2 and dump it to `br1` so that iVM2 can receive it. For an outbound packet, a SNAT rule will translate source IP 192.168.1.2 to 56.0.0.11. The source routing policy in rVM2 we discussed earlier will take over and forward it to the Internet.

Our network design offers high flexibility. Each rVM can host many iVMs with any number of floating IPs. Public/private traffic forwarding is not disrupted with iVM live migration, even across data centers. As a special case, if there is only one iVM demanding one floating IP on a rVM, we can directly associate the floating IP in the reality cloud onto that rVM's `eth0` to avoid internal forwarding on virtual public layer-2 network, thus reducing forwarding overhead. Moreover, our network design applies without changes to recursively nested inception clouds, as long as within the limit of physical resources.

4 Implementation

We have implemented a prototype inception cloud for continuous integration on top of a private IaaS cloud as our reality cloud. Both the inception and reality cloud use OpenStack as CMS. Specifically in the inception cloud, we run all OpenStack core components including compute, volume, image and identity services. Both the physical machines of the reality cloud and the rVMs run Ubuntu 12.04 LTS server, with a Linux kernel version 3.2.0-37. KVM with nested virtualization enabled [7] is the hypervisor.

We chose Open vSwitch version 1.9 to build the network of our inception cloud. The tunneling protocols supported by Open vSwitch include VXLAN and GRE. Because GRE packets are dropped by the default OpenStack firewall, we chose to use VXLAN which is tunneled in UDP. In principle, VXLAN uses UDP-based packet encapsulation with 24-bit virtual network identifier (VNI) supporting up to 16.7 million VNIs. We constructed a star-topology network for simplicity: each rVM builds a VXLAN tunnel with the gateway rVM, which acts as a hub for the virtual data center. We turned on the spanning tree protocol (STP) to avoid accidental network loops due to misconfiguration.

To automate the configuration and deployment process, we wrote Chef [2] cookbooks to setup both OpenStack and Open vSwitch VXLAN networks. Without any human intervention, we are able to launch an OpenStack-based CI inception cloud within minutes. It is worth mentioning that, CI inception clouds enable us to quickly deploy and evaluate OpenStack from any revision in the source code management system. After each set of testing, the CI inception cloud is easily and quickly torn down by destroying the rVMs.

In the preliminary evaluation, we mostly focus on functionality tests such as compute API correctness, network reachability, floating IP association, firewall effectiveness, live VM migration, etc. This is exactly the purpose for a CI cloud. Interestingly, we noticed some significant performance degradation in iVM compared to rVM during some I/O stress tests, and we plan to investigate further in future work.

5 Conclusion and Future Work

This paper presents inception, a nested IaaS cloud architecture. The idea originates from our experience in deploying and experimenting OpenStack with new features. We have been longing to lower the barrier of fast launching and running a large-scale customizable cloud, without worrying about dealing with physical bare metals. This aligns well with the use case of CI inception cloud outlined in Section 2. Far beyond that, inception clouds offer other major benefits against reality clouds, including hypervisor flexibility, app-specific resource allocation, and cross-provider wide-area deployment. We hope inception will lower the barrier of entrance to the IaaS cloud space and inspire further innovation from the research community.

Our future work are as follows: (1) Our inception clouds rely on layer-2 private/public networks. To improve network scalability, a more scalable control plane is necessary. Because Open vSwitch supports OpenFlow, we plan to build SDN controllers (e.g. similar to Onix [13]) to both eliminate ARP broadcast traffic and provide direct traffic routes (as opposed to go through the gateway as hub); (2) We plan to add new features into OpenStack, ranging from handling dynamic rVM addition/deletion, supporting extensible hypervisor functionalities [8, 14], and experimenting myriad resource scheduling/optimization strategies; (3) In addition to a private reality cloud environment, we plan to deploy inception clouds across multiple public providers; (4) We plan to investigate iVM I/O performance bottlenecks, and explore light-weight isolation technology such as Linux Containers as alternatives when applicable.

References

[1] Amazon Elastic Computing Cloud (EC2). `http:`

//aws.amazon.com/ec2/.

[2] Chef Opscode. http://www.opscode.com/chef/.

[3] Generic Routing Encapsulation (GRE). http://www.ietf.org/rfc/rfc2784.txt.

[4] OpenStack. http://openstack.org/.

[5] VMware vCloud Suite. http://www.vmware.com/.

[6] VXLAN A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-03.

[7] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: design and implementation of nested virtualization. In *OSDI*, 2010.

[8] 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 *EuroSys*, 2011.

[9] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI*, 2005.

[10] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: enabling intrusion analysis through virtual-machine logging and replay. In *OSDI*, 2002.

[11] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: harnessing memory redundancy in virtual machines. In *OSDI*, 2008.

[12] C. Kim, M. Caesar, and J. Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *SIGCOMM*, 2008.

[13] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *OSDI*, 2010.

[14] U. F. Minhas, S. Rajagopalan, B. Cully, A. Aboulnaga, K. Salem, and A. Warfield. Remusdb: Transparent high availability for database systems. *PVLDB*, 4(11):738–748, 2011.

[15] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[16] C. A. Waldspurger. Memory resource management in vmware esx server. In *OSDI*, 2002.

[17] D. Williams, E. Elnikety, M. Eldehiry, H. Jamjoom, H. Huang, and H. Weatherspoon. Unshackle the cloud! In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, HotCloud'11, 2011.

[18] D. Williams, H. Jamjoom, and H. Weatherspoon. The xen-blanket: virtualize once, run everywhere. In *EuroSys*, 2012.

[19] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *SOSP*, 2011.