In-Network Live Snapshot Service for Recovering Virtual Infrastructures

Ardalan Kangarlou and Dongyan Xu, Purdue University
Ulaş C. Kozat, Pradeep Padala, Bob Lantz, and Ken Igarashi, DOCOMO USA Laboratories

Abstract

Infrastructure as a Service (IaaS) has become an increasingly popular type of service for both private and public clouds. The virtual infrastructures that enable IaaS support multitenancy by multiplexing the computational resources of data centers and result in substantial reductions in operational costs. Since hardware and software failures occur on a routine basis in large-scale systems, it is imperative for cloud providers to offer various failure recovery options for distributed services hosted on such infrastructures. In this article we present *GENI-VIOLIN*, a new cloud capability that can checkpoint a stateful distributed service while incurring very low overhead. The unique aspect of GENI-VIOLIN compared to previous work is that GENI-VIOLIN exploits programmable OpenFlow switches to provide checkpointing services *in the network*, thereby requiring minimal changes to the end host virtualization framework. We have developed a prototype of GENI-VIOLIN using the GENI infrastructure, and have demonstrated GENI-VIOLIN's checkpoint and restore capability across multiple GENI sites.

Recent years have witnessed the emergence of virtualized public and private computing clouds that offer Infrastructure as a Service (IaaS). In this paradigm, cloud operators maintain the underlying physical infrastructure and provide virtual machines (VMs) to cloud users (e.g., Amazon EC2). Cloud users then deploy their applications or services within the VMs allocated to them. A virtualized infrastructure provides a flexible pay-as-you-go environment that allows scale out/in of services based on demand fluctuation. Real-life usage scenarios cover a wide spectrum including outsourcing of IT departments, providing new Internet services, and performing scientific computations. For many users, high availability (i.e., running applications/services when needed where needed) is a key proponent of cloud services. As hardware and software failures occur on a regular basis in large-scale environments [1, 2], IaaS providers should support primitives that enable failure recovery and service availability.

Current failure recovery mechanisms for cloud services have serious limitations. For many cloud providers, failure recovery entails replacing a failed VM with a new VM freshly booted from a custom VM image. This simple approach has two problems:

- Loss of runtime state
- Long setup time to bootstrap the VM, to reconfigure it, and to re-establish network connections

To capture and replay the runtime state, many techniques have been proposed that perform checkpointing at the application, operating system, or hypervisor level. A comprehensive comparison of techniques along with advantages and disadvantages of each has been presented in [3]. As cloud providers cannot make modifications to a VM's software stack to support checkpointing, checkpointing at the hypervisor level appears to be the most practical solution for IaaS systems.

Failure recovery mechanisms for VMs can be roughly classified into two main approaches. Fast failover is the technique used by high-availability systems like Remus [4], where the VM state is continuously synchronized to another physical host to create a backup copy of the VM in standby. After a VM failure, the standby copy can immediately take over and continue service without any loss or rollback of the execution state. In theory, one can run this mechanism for all VMs of a given service instance and attain continuous availability for the whole service. However, offering such a live replication service incurs high computational and bandwidth overheads, doubles the VM hosting costs, and is limited to a local area network. An alternative approach is to capture the global state of a cloud service using *distributed snapshots (checkpoints)*. This approach involves making modifications to either the virtual networking layer at the end hosts (e.g., the VNsnap system [3] on top of the VIOLIN [5] virtual networking layer) or the networking stack in the VM (e.g., checkpointing Emulab experiments [6]).

In this article, we propose GENI-VIOLIN, a distributed snapshot mechanism that operates exclusively at the network level and provides fault tolerance for stateful services provisioned over multiple VMs. The GENI-VIOLIN approach is applicable to a variety of virtualization platforms (e.g., VMware ESX, Xen, KVM) and only requires live VM migration support from the underlying virtualization platform. The unique aspect of GENI-VIOLIN that distinguishes it from the previous work is that GENI-VIOLIN exploits virtualizable and programmable networking hardware (e.g., OpenFlow switches [7]) to implement the distributed snapshot functionality. This capability makes GENI-VIOLIN one of the first

0890-8044/11/$25.00 © 2011 IEEE

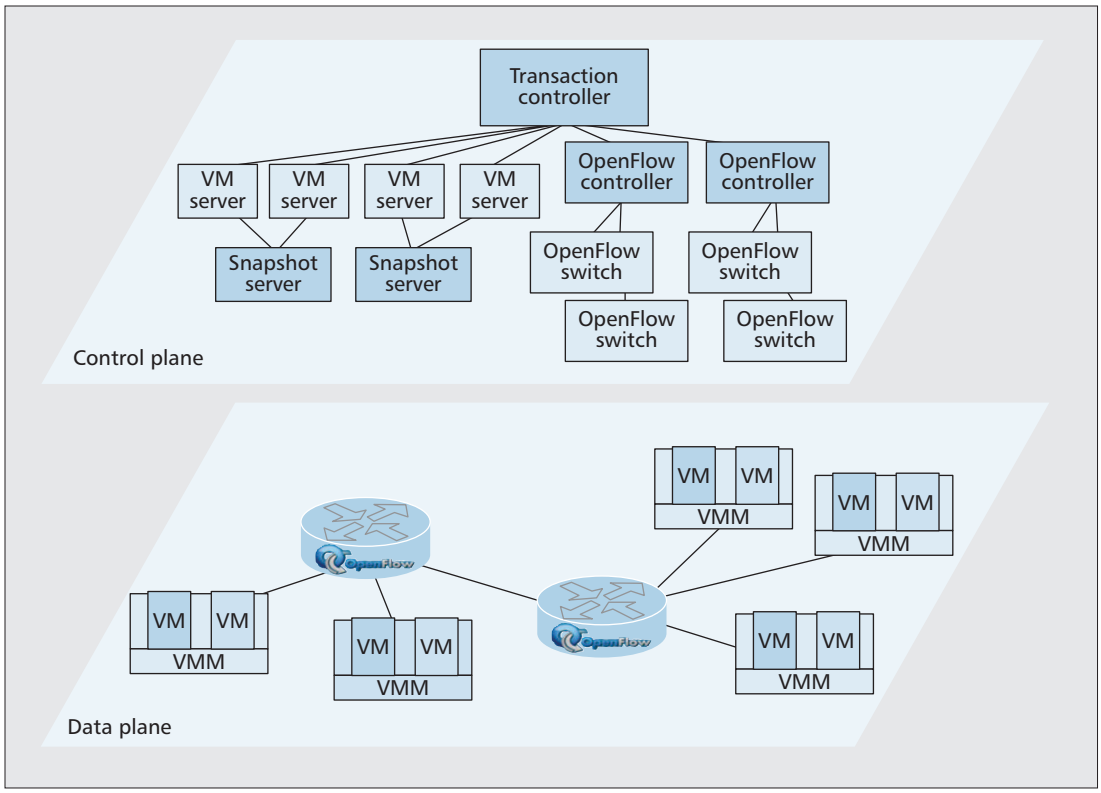

Figure 1. *To provide a live, distributed snapshot capability, GENI-VIOLIN adds several modules to the cloud's control plane: a transaction controller (TC) controls the operation of snapshot servers (SSs) to record state from off-the-shelf VM servers (VMSs), and custom OpenFlow controllers (OFCs) manage the flow tables of OpenFlow switches (OFSs). Besides using OFSs, no changes to the data plane are required, and snapshot taking only minimally perturbs VM computation and communication.*

systems of its kind to demonstrate a novel application of modern programmable networking hardware. GENI-VIOLIN's main features are summarized below:

- The distributed snapshot algorithm is moved into the network by exploiting OpenFlow switches.
- Moving the distributed snapshot functionality to the network enables a generic solution that requires minimal custom platform-specific modification to end hosts.
- GENI-VIOLIN can support snapshot operation for services that span multiple subnets, availability zones, or even different geographical regions.
- GENI-VIOLIN is completely transparent and requires no modifications to cloud application and system software.

We have built a prototype of GENI-VIOLIN on the GENI [8] testbed consisting of two geographically distributed data centers. Technical details of our implementation along with preliminary evaluation are presented in the rest of this article.

System Architecture

The GENI-VIOLIN architecture consists of multiple components that participate during different stages of the distributed snapshot operation. Figure 1 illustrates these components and their relationship to each other. For the remainder of this section, we briefly describe the role of each component within our proposed in-network snapshot solution.

VM Server

A VMS is a physical server that hosts VMs in the cloud data centers. GENI-VIOLIN requires only minimal changes to a VMS. Instead, it relies on standard functionalities such as live VM migration, VM pause, and VM resume that are commonly supported by different system virtualization platforms. Our prototype implementation is based on Xen [9] virtualization and operates similarly to the VM snapshot implementation in VNsnap [3]. Both GENI-VIOLIN and VNsnap rely on live VM migration to transfer the VM state to a remote host. It is only during the last iteration of migration that a VM is actually paused. Hence, both GENI-VIOLIN and VNsnap incur less than a second of downtime, since the VM is operational during much of the snapshot operation. Once the image is fully transferred, the VM migration is aborted, so the VM resumes operation on the same VMS. If the virtualization software provides a live VM checkpoint capability, one can leverage this functionality instead of relying on live VM migration. The current implementation of GENI-VIOLIN also relies on a signaling mechanism at the VMS, so other components of GENI-VIOLIN can be notified when a snapshot operation is in progress or complete. We cover the details of signaling more extensively later.

Snapshot Server

An SS serves as the destination for a live VM snapshot. A single SS can serve multiple VMSs, and a VMS can use different SSs at different times. An SS is responsible for:

- Receiving a VM image during snapshot and writing it to persistent storage
- Aborting the migration once a VM image is fully received so that the VM resumes operation on the same VMS
- Committing or aborting the written VM images based on the success of the snapshot operation
- Copying the VM images to other sites prior to snapshot restoration

Additionally, if feasible, one can move the VMS's signaling functionality entirely into the SS so that in-network snapshotting requires no changes to the VMS.

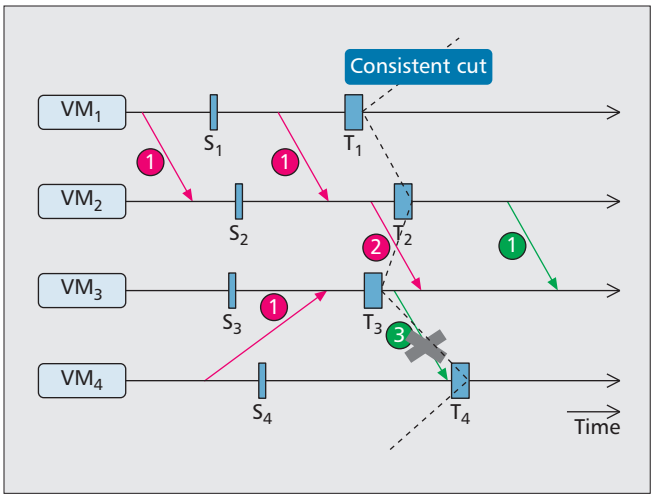

Figure 2. *An illustration of Mattern's snapshot algorithm [3] and handling of different types of messages. The snapshot of* VM_i *begins at time* S_i *and ends at* T_i.

Transaction Controller

A TC orchestrates the distributed snapshot operation and manages the interactions among different components of GENI-VIOLIN. A TC is aware of all the VMs that belong to a certain service and can initiate a distributed snapshot operation on demand or periodically. A TC's main responsibilities can be summarized as follows:

- It initiates VM migration from a VMS to an SS and monitors all the interactions between the VMS and the SS so that it knows whether a VM is in the pre-snapshot or post-snapshot state.
- Based on the snapshot state of the VM, the TC instructs the OFC to update routing configurations in the networking hardware (i.e., OpenFlow switches), so that consistency of the distributed snapshot state is guaranteed (to be discussed in the next section).
- The TC instructs an SS to commit VM snapshots when the snapshot operation is successful; otherwise, the VM snapshots are discarded.

OpenFlow Switch

Each OFS is an off-the-shelf switch compliant with the OpenFlow specification [7] and supports network virtualization in hardware. The key characteristic of an OpenFlow switch that distinguishes it from a traditional network switch is that the *flow table* in the switch can be managed by an OpenFlow Controller (OFC). Each flow can be defined based on a subset of header fields in the network packet. For example, a flow can be identified based on the VLAN ID, source/destination MAC address, source/destination IP address, or a combination of different fields. Additionally, for each flow in the flow table, there is an associated *action* that determines the fate of packets in that flow. A few examples of such actions are dropping packets, forwarding packets to a specific port, forwarding packets to the OFC (the default action), and packet encapsulation/decapsulation. These actions are determined by the OFC and are communicated to an OFS via the OpenFlow protocol.

OpenFlow Controller

An OFC manages the flow table of an OFS. GENI-VIOLIN incorporates a custom controller that updates flow tables based on the pre-snapshot or post-snapshot status of sender and destination VMs. More specifically, our custom OFC communicates with the TC during the snapshot operation and updates egress OpenFlow switches on the path from the source VM to destination VM to specify actions such as dropping, buffering, and injecting packets. In the next two sections, we briefly describe the distributed snapshot algorithm and then proceed with describing how these different components come together to provide an in-network, distributed, live snapshot service.

Mattern's Distributed Snapshot Algorithm

In a distributed system where VMs communicate with each other and maintain state as a result of that communication, some VMs may take longer than others to record a checkpoint. As a result, it is critical to preserve causal consistency during the distributed snapshot operation so that the global snapshot can be safely restored. In this light, the sole purpose of a snapshot algorithm is to guarantee such consistency despite the asynchronous nature of distributed snapshots. Over the years different distributed snapshot solutions have been proposed in the literature [3, 10–13]. VNsnap [3] is one such system that demonstrates applicability of Mattern's snapshot algorithm [14] to the VIOLIN virtual networked infrastructure [5]. To enable taking snapshots of unmodified VMs, VNsnap implemented this distributed snapshot algorithm within the virtual networking layer of a VMS.

Similar to VNsnap, GENI-VIOLIN adopts Mattern's distributed snapshot algorithm [14] to achieve a consistent, global snapshot. However, by leveraging programmability of OpenFlow networking hardware, GENI-VIOLIN moves the implementation of the distributed snapshot algorithm from the VMSs into the network. In this section, we only briefly describe Mattern's snapshot algorithm to set up the background for the GENI-VIOLIN implementation in the next section. We therefore encourage readers to refer to [3] for a more thorough discussion of the snapshot algorithm and the proof of its applicability.

Figure 2 presents scenarios that can arise during an asynchronous system snapshot and shows the way Mattern's snapshot algorithm copes with each. This figure shows a service composed of four communicating VMs. The TC instructs each virtual machine VM_i to start a snapshot at time S_i, and the VMs complete their snapshot at time T_i. As described earlier, just before T_i, VM_i is paused, and its final memory state is transferred to a snapshot server (SS) as the snapshot image. Based on this figure, VM_i can be in one of the three possible states at a given time t:

- Pre-snapshot, where the VM did not start the snapshot operation (i.e., $t < S_i$)
- Snapshot, where the VM has started the snapshot operation but not completed it (i.e., $S_i \le t < T_i$)
- Post-snapshot, where the VM has finished the snapshot (i.e., $t \ge T_i$)

Additionally, the packets exchanged between VMs can be classified into one of three types:

- Type-1: These packets are sent from VMs in pre-snapshot or snapshot states to VMs in pre-snapshot or snapshot states. Also, packets from post-snapshot VMs to other post-snapshot VMs are Type-1 packets.
- Type-2: These packets are sent from VMs in pre-snapshot or snapshot state to VMs in post-snapshot state.
- Type-3: These packets are sent from VMs in post-snapshot state to VMs in the pre-snapshot or snapshot state.

In Fig. 2, Type-1, Type-2, and Type-3 packets are labeled as 1, 2, and 3 respectively. A close examination of each type reveals the following: Type-1 packets do not violate consistency of distributed snapshots as the snapshot image of both the sender and receiver VMs agree whether a packet was sent and received before snapshot (e.g., a packet from a pre-

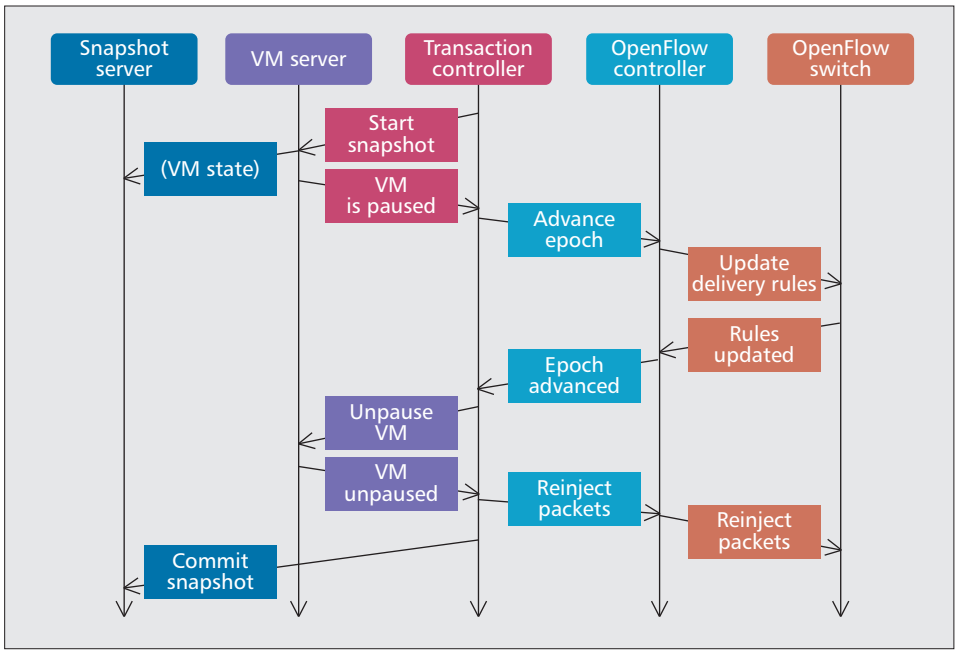

Figure 3. *Interactions between different components of the GENI-VIOLIN architecture.*

snapshot/snapshot VM to another pre-snapshot/ snapshot VM) or not (e.g., a packet from a post-snapshot VM to another post-snapshot VM). Type-2 packets are only included in the snapshot of sending VMs. Thus, when the system is restored from a given snapshot, Type-2 packets result in a scenario that is equivalent to the case where these packets are lost in the network. Hence, these packets do not result in any conflicting state between sender and receiver VMs.

Type-3 packets, on the other hand, violate causal consistency and should not be included in the distributed snapshot. To illustrate this point, we can assume the Type-3 packet in Fig. 2 is included in the global snapshot. As a result, when the distributed snapshot is restored, VM_4's snapshot state indicates receiving a packet from VM_3 whereas VM_3's snapshot has no notion of sending that packet. Delivering Type-3 packets results in either discarding such packets and stalling a connection (e.g., for TCP connections) or duplicate processing (e.g., for UDP connections) after snapshot restoration.

To preserve consistency and avoid the two undesirable scenarios just described, Type-3 packets should not be delivered to destination VMs before their snapshot is finalized. One way to ensure that is to simply drop Type-3 packets in the network. This, however, could incur a significant loss rate (for UDP connections) or transmission slowdown (for TCP connections) depending on the rate of communication and the degree of skew between the snapshot completion times of the VMs. While our live VM snapshot implementation reduces individual VM downtime during the snapshot operation to less than a second, dropping Type-3 packets would result in repetitive bursty losses and transmission slowdowns that can last a few seconds. To alleviate this problem, one option is to hold and buffer Type-3 packets in the network and inject them as soon as the destination VM completes its snapshot. Our GENI-VIOLIN implementation supports both options.

GENI-VIOLIN Snapshot Algorithm Implementation

In the GENI-VIOLIN implementation, the distributed snapshot algorithm is implemented in the network via the OpenFlow controller (OFC) and OpenFlow switches (OFSs). Figure 3 illustrates the interactions among different components of the GENI-VIOLIN architecture to achieve a causally consistent snapshot image. The interactions during the snapshot operation can be summarized as follows:

1 The transaction controller (TC) starts the snapshot operation for a service by sending a `START_SNAPSHOT` message to all the VMSs that host VMs constituting that service.
2 Upon receiving a `START_SNAPSHOT` message, a VMS starts transferring the VM state to an SS. At the final stage of transfer (i.e., the last iteration of migration) where the VM is paused and all dirty pages are transmitted to the SS, the VMS informs the TC by sending a `VM_PAUSED` message.
3 After receiving `VM_PAUSED`, the TC instructs the OFC via an `ADVANCE_EPOCH` message to update the flow table in the egress OFS so that the packet delivery rules for a VM reflect the snapshot algorithm described earlier (more details later).
4 Once the OFC finishes updating the OFS for a particular VM, it notifies the TC to unpause the VM with a `VM_UNPAUSE` message. This message in turn results in the TC sending a request to the VMS to unpause the VM and a `REINJECT_PACKETS` request to the OFC so that the buffered packets can be injected. Additionally, based on the success or failure of the snapshot operation, the TC instructs the SS to commit or abort the snapshot images.

Hereafter, we focus on the main contribution of GENI-VIOLIN: the use of programmable networking hardware to implement the snapshot algorithm in the network. As alluded to earlier, packet forwarding and the snapshot algorithm in GENI-VIOLIN are done via OFSs under the control of the OFC. While our design can support multiple OFCs and service domains, for the sake of ease in description we assume one OFC manages all OFSs that provide network connectivity for one service. Figure 4 presents simplified pseudocode for the OFC, and captures the sequence of steps that take place when a service is started and when its snapshot is taken.

Initially, when a service is started, the OFC and OFSs have no state about the VMs that make up the service. Therefore, when VM_i sends a packet, the egress OFS sends a `PACKET_IN` message to the OFC to query how it should treat that packet. This message prompts the OFC to update the flow table of

```
def init():
        for vm in vms:
                post_snapshot[vm] = False
        for switch in switches:
                switch.delete_all_forwarding_rules ()

def packet_in (switch, inport, src, dst, packet):
        if is_access_port (switch, inport):
                egress_switch[inport] = src
        if post_snapshot (src) and not post_snapshot(dst):
                buffer[dst].append ((packet, switch))
        else:
                switch.install_forwarding_rule (src, dst)
                switch.forward (packet)

def advance_epoch (src):
        post_snapshot[src] = True
        egress_switch[src].delete_forwarding_rules_from (src)

def reinject_packets (dst):
        assert (post_snapshot[dst])
        for packet, switch in buffer[dst];
                switch.forward (packet)
        buffer[dst] = {}
```

Table 1. *Simplified pseudocode for the OFC.*

the egress OFS so that VM_i's packets can be forwarded. This message also sets VM_i's state in the OFC to *pre-snapshot*. During the course of the snapshot operation, when the OFC receives the ADVANCE_EPOCH message, the OFC sets VM_i's state to *post-snapshot* and deletes all forwarding rules for VM_i in the egress OFS. The removal of the forwarding rules again prompts outgoing packets from VM_i to trigger PACKET_IN message to the OFC. At this moment, the OFC can enforce the snapshot algorithm by not allowing forwarding of packets from a post-snapshot VM to a pre-snapshot VM (i.e., Type-3 packets). More precisely, the OFC instructs the OFS to buffer Type-3 packets and inject them back in the network when the receiver VM transitions to the post-snapshot state. Since the delivery of Type-1 and Type-2 packets does not violate the consistency of the global snapshot, the OFC can safely insert forwarding rules upon receiving PACKET_IN messages for these type of packets.

GENI-VIOLIN Evaluation

This section describes two sets of experiments involving GENI-VIOLIN. We demonstrate applicability of GENI-VIOLIN to taking a snapshot of a GENI slice that runs a distributed MPI-based application. Additionally, we present a few microbenchmark results on the performance of GENI-VIOLIN.

POV-Ray Distributed Snapshot

We have evaluated GENI-VIOLIN using two GENI sites, one at the University of Utah (ProtoGENI Utah) and the other one at the GENI Project Office (ProtoGENI GPO) in Cambridge, Massachusetts. At each site, the hardware for our experiments consisted of eight Dell servers with one 2.4 GHz 64-bit Quad Core Xeon, 12 Gbytes of RAM, and six Broadcom NetXtreme II BCM5709 GbE network interface cards (NICs) connected via an OFS. Figure 5 illustrates our setup at both sites. We dedicated four servers as VM Servers (VMS) at each site (Utah-A, Utah-B, Utah-C, and Utah-D at ProtoGENI Utah, and GPO-A, GPO-B, GPO-C, and GPO-D at ProtoGENI GPO); each VMS runs one VM with 256 Mbytes of RAM. We used two servers at ProtoGENI Utah as SSs, denoted Utah-E and Utah-F in Fig. 5. Each SS serves two VMSs in our configuration. Additionally, we used one server for running the transaction controller (TC) and the GENI-VIOLIN user interface (UI) and one for running the OFC. The two GENI sites were connected using an Internet2 link.

Our experiment involved taking a snapshot of POV-Ray [15], a distributed MPI-based rendering application, at ProtoGENI Utah (Fig. 5a), simulating a hardware failure at ProtoGENI Utah by abruptly killing all four VMs running POV-Ray (Fig. 5b), and restoring the snapshot at ProtoGENI GPO (Fig. 5c). Figure 5 also demonstrates various aspects of the GENI-VIOLIN operation, such as the GENI-VIOLIN UI, the output of POV-Ray (the rendered bonsais) when the snapshot is taken and restored, the network activity on the SS during the snapshot operation (spikes at Utah-E and Utah-F), and the network activity on the Internet2 link when the distributed snapshot is transferred from ProtoGENI Utah to ProtoGENI GPO. GENI-VIOLIN was successfully demonstrated at the 9th GENI Engineering Conference (GEC-9) in November 2010, and a demo of GENI-VIOLIN can be viewed at [16].

The measurements we obtained suggest that on average it took 10 s to snapshot a 256-Mbyte VM running POV-Ray when one SS is serving two VMSs. However, the VM downtime during the snapshot did not exceed 200 ms. Our measurements show that on average it took 28 ms for the OFC to update the flow table in the OFS and notify other components. It also took slightly more than a minute to transfer the four VM snapshot images from ProtoGENI Utah to ProtoGENI GPO across the Internet2 link.

Microbenchmark Experiments

In this section, we describe some microbenchmark experiments to illustrate the effectiveness and limitations of packet buffering. For these experiments, our testbed environment consisted of several HP Proliant BL460c G6 blades, one NEC IP8800/S3640-48T2WX OpenFlow switch with 48 1-Gb/s interfaces, and four Lenovo ThinkPad X200s laptops. All the blades and laptops are directly connected to the OpenFlow switch through standard Ethernet cables or USB-Ethernet interfaces. The laptops host VMs that run *Iperf* [17], and the blades run the OFC, TC, and SS.

Recall earlier we said that for asynchronous distributed snapshots, dropping Type-3 packets results in transmission slowdown for TCP packets. This overhead can be particularly significant when there is a large skew between completion times of VM snapshots. For the experiments in this section, we study scenarios where we have two VMs and the sending VM completes its snapshot a few seconds before the receiving VM. As a result, all the packets transmitted by the sender during the snapshot skew period become Type-3 packets. Figure 6b shows multiple network traces obtained through *Tcpdump* when we take a snapshot 2 s into the experiment and set the length of skew to 1, 2, and 4 s as represented by red, green, and blue traces, respectively. Comparing Figs. 6a and 6b suggests that buffering packets at the switch greatly reduces the length of TCP backoff where the sender exponentially reduces its transmission rate until it receives an acknowledgment from the receiver. When buffering is enabled, the receiving VM does not have to wait for a TCP timeout and retransmission to receive the Type-3 packets, which results in the reduced overhead (i.e., the shorter length for the flat "no-progress" period in the network trace) shown in Fig. 6b. Our experiments with UDP transfers at low rates also suggested that packet buffering and injection helps reduce packet loss during a snapshot.

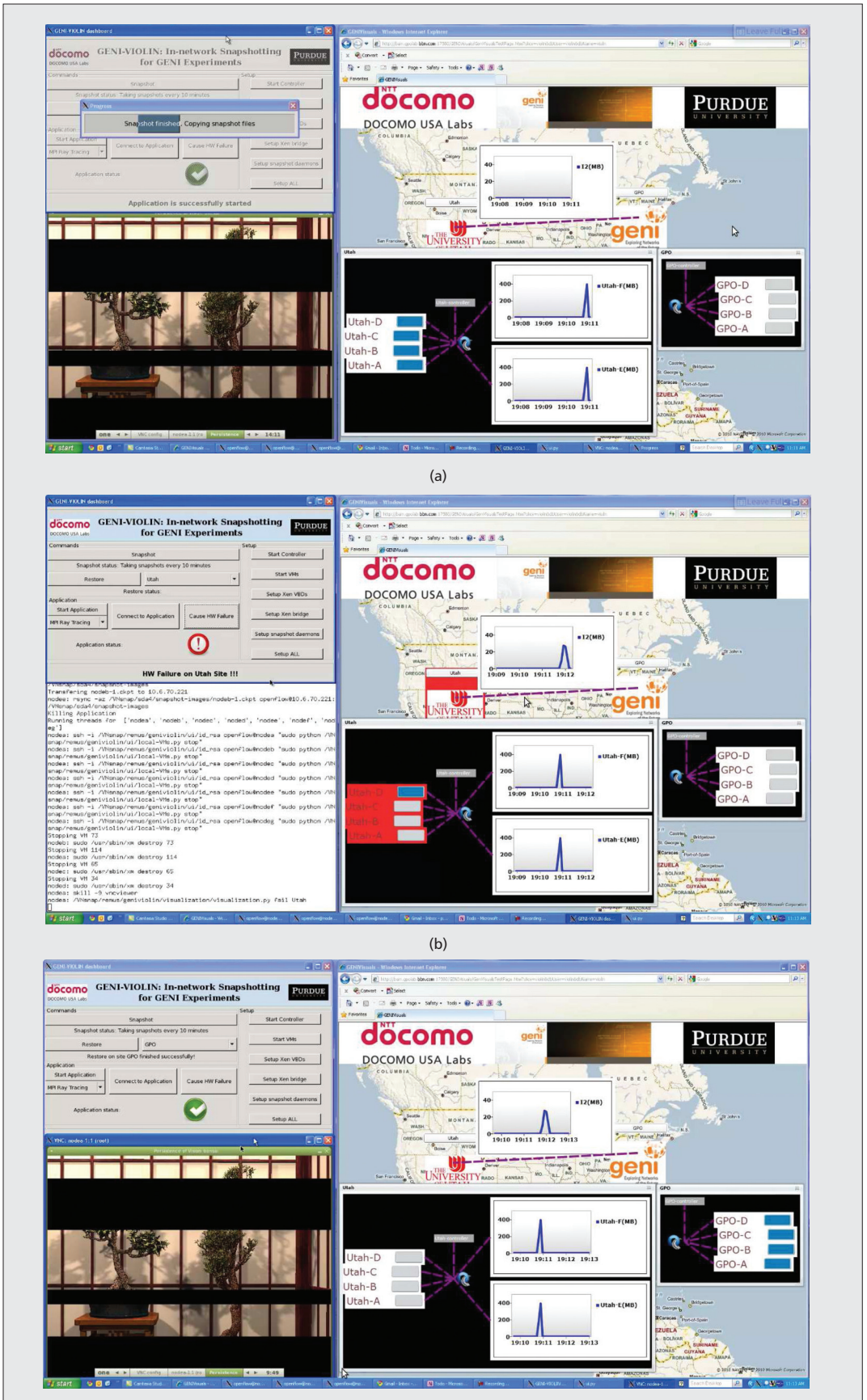

Figure 5. *A sequence of screenshots displaying the operations of GENI-VIOLIN: a) a snapshot of POV-Ray at ProtoGENI Utah; b) hardware failure at ProtoGENI Utah; c) snapshot restoration at ProtoGENI GPO.*

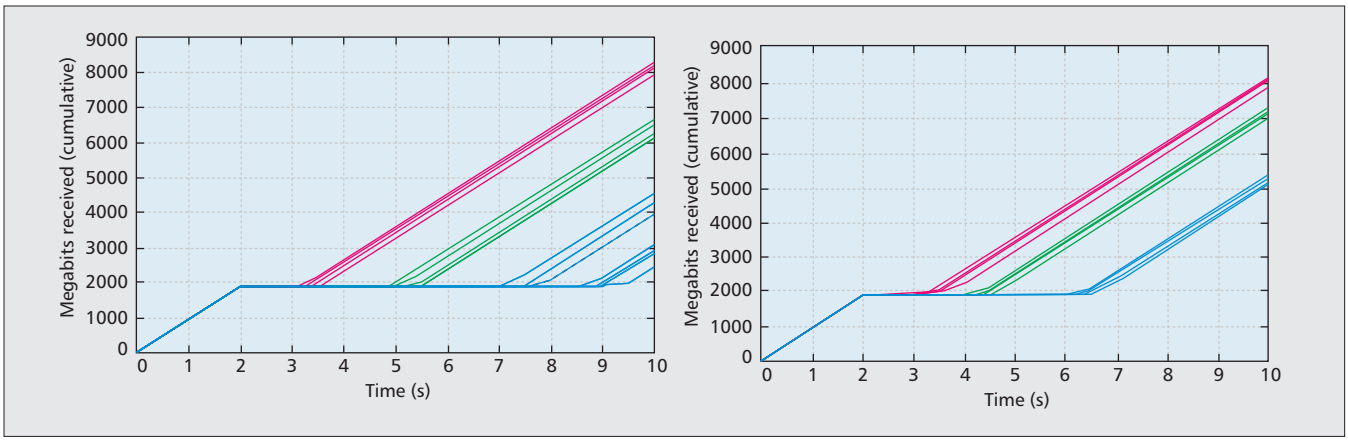

Figure 6. *Effectiveness of buffering in reducing TCP backoff: a) dropping Type-3 packets; b) buffering and injecting Type-3 packets at the switch.*

Discussion

This section focuses on three aspects of GENI-VIOLIN that require further study and improvement. First, as discussed earlier, the skew in completion time of VM snapshots can potentially result in many buffered packets. For homogeneous setups where all VMs have the same memory size, the skew is usually not significant. However, in heterogeneous setups where VMs can have different memory allocations, it is inevitable that some VMs complete their snapshots much earlier than others. Since the VM size, rather than the application workload, is the main determiner of the length of the snapshot operation for a given VM, one way to alleviate this problem is to profile the duration of snapshots for VMs of different sizes. Using this profile information, GENI-VIOLIN could then delay starting snapshots for smaller VMs so that all VMs would finish their snapshots at about the same time.

Second, one direct consequence of asynchrony in completion time of snapshots is the buffering that takes place in GENI-VIOLIN. While delaying the start of snapshot for smaller VMs could reduce the skew and buffering requirements, it is unlikely that buffering can be eliminated. The main issue with buffering is its overhead in terms of the number of `PACKET_IN` and `PACKET_OUT` messages sent to/from the OFC and the memory requirement to store the buffered packets. Our initial investigation suggests that for very low transfer rates (e.g., 1 Mb/s), buffering can be done at the OFS without any problems. However, for higher sustained transfer rates (e.g., 10 Mb/s and higher) resources at the switch and controller can quickly become exhausted. The problem is further exacerbated by transport protocols such as UDP that do not employ flow or congestion control. We are currently investigating alternative methods of buffering by performing buffering and injection using middleboxes (e.g., dedicated servers for buffering) that can store a large number of packets and process them at line rate. We believe such middleboxes along with reasonable ratios of OFCs relative to OFS's and SS's relative to VMS's can result in a very scalable solution.

Third, GENI-VIOLIN's unique in-network snapshot capability allows capturing the network state (e.g., the flow tables of switches, routers, etc.) in addition to capturing the CPU, memory, and storage states. Such saved state could be used by the OFC during snapshot restoration to proactively set up flow tables in an OFS. This capability could result in preserving the network topology of a service (e.g., the topology of a networking experiment in the cloud), resulting in lower overhead compared to reactive flow table setup during snapshot restoration.

Conclusion

We present GENI-VIOLIN, an in-network live distributed snapshot solution for services hosted over public and private clouds. GENI-VIOLIN differs from previous distributed snapshot solutions as it relies on the capabilities of modern networking hardware. Given that the snapshot algorithm is implemented entirely using programmable networking hardware, GENI-VIOLIN requires minimal modifications to VM servers irrespective of the platform virtualization technology used by the cloud provider. Our prototype implementation and evaluation results using real-world distributed infrastructure and applications demonstrate the suitability of GENI-VIOLIN for services that primarily use TCP connections. We also propose alternative implementations to reduce the overhead of GENI-VIOLIN and improve its applicability to a wider range of cloud services.

References

[1] J. Dean, "Underneath the Covers at Google: Current Systems and Future Directions," 2008 Google I/O, San Francisco, CA, May 2008, http://sites.google.com/site/io/underneath-the-covers-at-google-current-systems-and-future-directions.
[2] G. DeCandia *et al.*, "Dynamo: Amazon's Highly Available Key-value Store," *Proc. SOSP'07*, Stevenson, Washington, Oc. 2007.
[3] A. Kangarlou, P. Eugster, and D. Xu, "VNsnap: Taking Snapshots of Virtual Networked Environments with Minimal Downtime," *Proc. 39th IEEE/IFIP Int'l. Conf. Dependable Systems and Networks (DSN-DCCS 2009)*, Estoril, Portugal, June 2009.
[4] B. Cully *et al.*, "Remus: High Availability via Asynchronous Virtual Machine Replication," *Proc. NSDI'08*, San Francisco, CA, Apr. 2008.
[5] X. Jiang and D. Xu, "VIOLIN: Virtual Internetworking on Overlay INfrastructure," Technical Report CSD TR 03-027, Purdue University, 2003.
[6] A. Burtsev *et al.*, "Transparent Checkpoints of Closed Distributed Systems in Emulab," *ACM EuroSys 2009*.
[7] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comp. Commun. Review*, Mar. 2008.
[8] The Global Environment for Network Innovations (GENI), http://www.geni.net/.
[9] P. Barham *et al.*, "Xen and the Art of Virtualization," *ACM SOSP*, 2003.
[10] O. Laadan, D. Phung, and J. Nieh, "Transparent Checkpoint/Restart of Distributed Applications on Commodity Clusters," *IEEE Int'l. Conf. Cluster Computing*, 2005.
[11] J. F. Ruscio, M. A. Heffner, and S. Varadarajan, "DejaVu: Transparent User-Level Checkpointing, Migration, and Recovery for Distributed Systems," *IPDPS 2007*.
[12] S. Sankaran *et al.*, "The LAM/MPI Checkpoint/Restart Framework: System Initiated Checkpointing," *Proc. LACSI Symp.*, Sante Fe, 2003.
[13] D. P. Scarpazza *et al.*, "Transparent System-Level Migration of PGAS Applications using Xen on Infiniband," *Proc. IEEE Int'l. Conf. Cluster Computing*, 2007.
[14] F. Mattern, "Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation," *J. Parallel and Distributed Computing*, 1993.
[15] Persistence of Vision Raytracer (POV-Ray), http://www.povray.org/.
[16] GENI-VIOLIN GEC9 Demo, http://vimeo.com/16535013/.
[17] Iperf, http://sourceforge.net/projects/iperf/.

Biographies

ARDALAN KANGARLOU [M] (ardalan@cs.purdue.edu) is a Ph.D. candidate in the Department of Computer Science at Purdue University. He received a B.S. degree from the University of Southern Mississippi in 2004 and an M.S. in computer science from Purdue University in 2006. His current research areas include virtualization, cloud computing, and distributed systems. He is a member of the ACM and USENIX.

ULAS C. KOZAT [SM] (kozat@docomolabs-usa.com) is currently serving as principal researcher at DOCOMO USA Laboratories, Palo Alto, California, where he heads the network architecture team. He received his Ph.D., M.S., and B.S. degrees, all in electrical engineering from the University of Maryland, College Park, George Washington University, Washington, DC, and Bilkent University, Ankara, Turkey, in 2004, 1999, and 1997, respectively. He worked at HRL Laboratories and Telcordia Technologies Applied Research as a research intern.

PRADEEP PADALA (ppadala@docomolabs-usa.com) is a research engineer at DOCOMO USA Laboratories working on virtualization and cloud computing with a special focus on mobile computing. He received his B.E. from the National Institute of Technology, Allahabad, India, his M.S. from the University of Florida, and his Ph.D. from the University of Michigan in 2000, 2003, and 2009, respectively. He received the best paper award at the USENIX Annual Technical Conference in 2010. More about his research can be found at http://ppadala.net.

BOB LANTZ (rlantz@cs.stanford.edu) founded the Mininet (openflow.org/mininet) project for rapid prototyping of software-defined networks, and encourages everyone to experiment with it. In addition to work on GENI-VIOLIN at DOCOMO Laboratories, he has contributed to OpenFlow and related efforts at Stanford and Arista Networks. He completed his Ph.D. at Stanford in 2007, developing Parallel SimOS, a full-system simulator for large multiprocessors.

KEN IGARASHI (igarashi@docomolabs-usa.com) has been with NTT DOCOMO since 2000 and with DOCOMO USA Laboratories, Palo Alto, California, since 2007. He has expertise in cloud computing, server virtualization, noSQL database systems, and TCP/IP.

DONGYAN XU (dxu@cs.purdue.edu) is an associate professor of computer science and electrical and computer engineering (by courtesy) at Purdue University. He received a B.S. degree from Zhongshan (Sun Yat-Sen) University in 1994 and a Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 2001. His current research areas include virtualization technologies, computer malware defense, and cloud computing. He is a recipient of a U.S. National Science Foundation CAREER Award.