Towards an Experimental LegoLand: Slice Modification and Recovery in ExoGENI Testbed

Yufeng Xin[1](✉), Ilya Baldin[1], Anirban Mandal[1], Paul Ruth[1], and Jeff Chase[2]

[1] RENCI, University of North Carolina at Chapel Hill, Chapel Hill, NC 27517, USA
yxin@renci.org

[2] Department of Computer Science, Duke University, Durham, NC 27708, USA

Abstract. This paper describes advanced capabilities that were deployed recently in the ExoGENI testbed to offer increased flexibility in provisioning, modifying, and recovering the topologies and the configuration settings of the virtual systems, or *slices*, in which experiments are run. Using the analogy of building complex structures with LEGO blocks, we envision an environment in which users arbitrarily scale out, scale in, scale up, and scale down their topologies using various modular constructs of compute, storage, and network resources. Portions of topologies can be shut down and brought back up to support resiliency, repeatability, migration, and other needs of the control software or application. Distributed applications running inside of slices can require programmatic control over the evolution of the topology as the execution progresses. The introduced capabilities, *slice modification* and *slice recovery*, are used either with the user GUI or through the programmable APIs. These new features expand the range and ease of options available to cloud-control software and to application developers as they test their designs at scale.

1 Introduction

The work that we present here endows researchers in networking and in distributed computing with flexible, efficient control over the scaling and configuration of network topologies in real time. It represents a substantive improvement in their ability to test the performance, scalability, and resiliency of the systems under investigation. Our work equips networked applications that are deployed inside and between data centers and clouds to operate more nimbly and robustly, optimizing available compute, storage, and networking resources. These types of requirements have been studied extensively as distributed application technologies and their uses have evolved [6,9,10].

Modern testbeds in networking and in distributed systems support virtual machines (VM) and other virtualization mechanisms [12,13] not only to improve usability and configurability but also to optimize resource utilization. Thus, they

This work is supported by the US National Science Foundation through the GENI initiative and NSF awards ACI-1245926, ACI-1440715, and CNS-1329745.

DOI: 10.1007/978-3-319-49580-4_4

resemble the public clouds deployed by Amazon, RackSpace, and other commercial providers. GENI, the federation of multiple testbeds, is consequently evolving into a cloud-based infrastructure-as-a-service (IaaS) system suited to real-world deployments of innovative distributed services and Future-Internet architectures as well as to experimentation [5].

Individual virtual machines provide a great deal of flexibility in configuring the performance attributes of individual compute nodes [14]. Allowing generalized manipulation of the topology interconnecting them, however, remains an open challenge. The only exceptions are a few typical data-intensive or web-service applications within a cloud in which the size of the worker server pool is allowed to be adjusted [10]. The challenge is greater in the case of a large-scale federated testbed system such as GENI, which includes multiple cloud sites connected by multi-domain networks. Each topology-modification action may involve modifying, provisioning, or releasing resources across multiple providers and may require provisioning of inter-site networking.

GENI creates virtual topologies for users. Each topology is called a *slice*. Individually configurable indivisible elements of slice topology are called *slivers*. Slivers can be individual virtual machines, bare-metal nodes, or links acquired from transit providers. Throughout this paper, we will use the term "slice"; however, instead of "sliver", we prefer *reservation* because we always associate a start and an end time with each sliver.

In this paper, we address four types of *topology-scaling*, defined as follows: (1) *Scale up*: increasing the size of a virtual cluster by adding more homogeneous nodes, or by raising the bandwidth of the network link in a slice; (2) *Scale down*: decreasing the size of a virtual cluster by removing nodes, or by reducing the bandwidth of reserved network links in a slice; (3) *Scale out*: adding new links, nodes, clusters, or third-party resources and linking them to existing reservations; (4) *Scale in*: deleting existing links, nodes, clusters, or third-party resources. We further define *slice modification* to be some form of *topology scaling* associated with reservation *term extension*. Reservation extension is required to ensure that the topology continues to exist after slice modification and to prevent older reservations from expiring. *Slice recovery* allows the system to stop, and then to restart, a slice while preserving its topology and resource accounting in the IaaS control system.

We focus on the implementation of generalized slice-modification capabilities in the ExoGENI testbed [3,7]. Since its inception seven years ago, ExoGENI has become an ambitious multi-domain IaaS system that has evolved well beyond the original design goal of a built-to-order virtual networking testbed. It is powered by a suite of information models, topology abstractions, and embedding algorithms, with resource-orchestration and -control software. It includes experimental tools that orchestrate a federation of independent cloud sites and networking providers through their native IaaS interfaces [1,3]. Due to the architectural flexibility of the ORCA [8,11] control software, ExoGENI continues to evolve. It now possesses the comprehensive generalized topology-scaling capabilities described here, which distinguish it from other similar testbeds.

The remainder of the paper presents our major contributions in three parts. In Sect. 2, we review the key components of ORCA, including system information models, topology abstraction, inter-domain orchestration, and end-to-end provisioning. We also describe the details of slice modification and recovery functions. In Sect. 3, we demonstrate several use cases that benefit from these new capabilities. Section 4 briefly states conclusions and ongoing work.

2 Slice Modification and Control Framework Design

In this section, we first motivate the slice-topology scaling with two typical use cases in cloud application and networking experiments. We then explain the technical challenges and the software enhancements that we added to the ORCA control framework to enable the slice modification. Due to the modular nature and flexible resource-description mechanisms of ORCA, most of the modifications were done outside of the ORCA core code, which handles such basic functions as agent communication, state maintenance, and recovery. We did modify the ORCA state machines, however, to support the sliver-modify cycle, which is discussed later in this section.

2.1 Motivating Examples

Figure 1 shows two types of virtual clusters that are commonly requested by big-data or tiered web-service applications. A virtual cluster request abstraction can be represented as N (virtual) nodes connecting to a virtual root R via links with bandwidth B as shown in Fig. 1(a). A more general representation is a virtual oversubscribed cluster [4] as shown in Fig. 1(c). Here, N VMs are grouped in a two-layered structure, where V_i is the virtual switch for connecting VMs in the group C_i with intra-group and inter-group bandwidths $B_i^{'}$ and B_i, respectively. A cluster is expressed by *NodeGroup* abstraction in ExoGENI, which specifies a group of VMs of identical configuration and connectivity.

The sizes and locations of the node groups are the most important scaling factors affecting the performance. A simple virtual cluster can be *scaled up* by

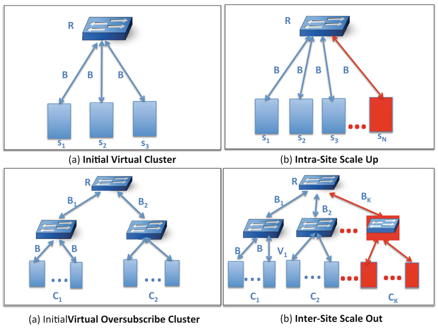

Fig. 1. Virtual cluster topology scaling

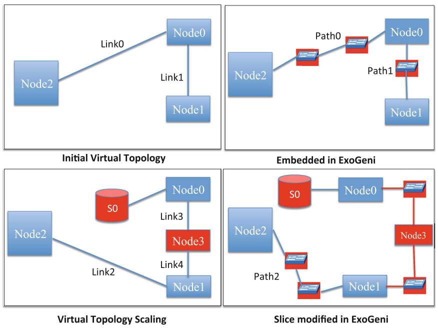

Fig. 2. General topology scaling

adding more nodes, or *scaled down* by deleting the existing nodes, as shown in Fig. 1(b). In this case, the entire cluster is embedded in a single cloud site. The modification action reduces to attaching/deleting the NICs of the newly added/deleted VMs to/from the broadcast link, which is a VLAN that is instantiated in the site switch. The case becomes much more complex when an oversubscribed virtual cluster consists of multiple sub-clusters that are embedded in different cloud sites. As shown in Fig. 1(d), we call this type of modification a *scale-out*, since the new sub-clusters are added from other sites. This mechanism is enabled by an *Exchange Domain* [15] deployed in ExoGENI. It can create multi-site broadcast domains to support large-scale distributed virtual clusters.

Figure 2 shows a sequence of modifications in a generic slice. The initial request is for a simple bus topology of three network nodes, as shown in Fig. 2(a). The topology is created in the testbed on an inter-site path (Link0 in Path0) and on an intra-site path (Link1 in Path1); the manifest topology is shown in Fig. 2(b). Next, modifications are made to the virtual topology, as shown in Fig. 2(c). In the example, the system deleted both Path0 and Path1 (*scale-in*), added a storage device to Node0, created a new link between Node2 and Node1, and created another new path between Node0 and Node1 via a new node, Node3 (*e.g.*, a router) (scale-out). The resulting topology is depicted in Fig. 2(d).

In addition to the basic topology-scaling performance, we can easily see the value of topology modification in other important system studies. For example, the option of adding or deleting arbitrary links in the virtual topology would be very useful in studying fault tolerance or congestion-avoidance performance. Link and node deletion can emulate network link or node failures and can be used to exercise backup and rerouting strategies. The ability to add new components to the topology, especially new types of resources (*e.g.*, storage nodes on demand), would yield significant benefits for distributed or big-data system designs.

2.2 Control Framework Support

We implemented slice modification in the ORCA control framework using extensions to resource-control policies and substrate drivers, all pluggable components in ORCA architecture. These modifications consisted of two parts. First, there were modifications to ORCA core-state machines to support a variety of atomic *reservation modify* operations. Examples of these include inserting new SSH keys into a compute node, adding or removing network interfaces from a compute node, and changing the bandwidth of a link. Second, we modified the control plugins to help orchestrate *slice modify* operations as sequenced sets of operations that *created* new reservations, *removed* or *modified* existing ones. This function required reservation modifications to accommodate stitching or unstitching parts of the slice. For instance, linking a new node to an existing node in the topology (*i.e.*, a scale-out) requires creating new link and node reservations as well as modifying the reservation of the existing node to add a new network interface attaching it to the new link, as shown in Fig. 2(d).

In this paper, we focus on the enhancements to ORCA that were created to support slice modification. The sequencing of provisioning operations in ORCA is part of its core functionality and is described more fully in [2,3], while further details of ExoGENI architecture and ORCA implementation can be found in [1] and on the project website [7].

A critical design principle in ORCA is that of maintaining provider autonomy, in which substrate providers advertise topology information and resource counts at the desired level of granularity and commitment. This feature allows providers to decide whether to under - or oversubscribe their resources and enables them to maintain the privacy of their internal topologies. The abstract representations of provider topology and resource levels are delegated to a broker in the form of resource pools of different types (VLAN, VM, etc.) with associated constraints on total bandwidth, core counts, available memory, and storage space.

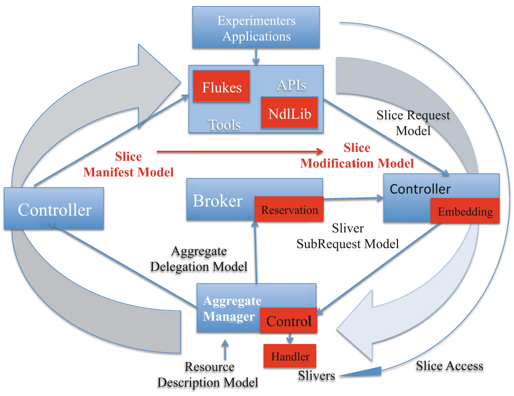

Fig. 3. ORCA information model and slice lifecycle

An ORCA user agent called *controller* queries the broker in real time for available resources, then constructs the inter-site topology from individual abstractions. Next, it embeds the user topology request and generates individual resource reservations for different aggregate managers (AMs) representing resource providers. These reservations are sent to the AMs to be redeemed for actual resources, which are provisioned by the respective AMs.

Each AM has its own resource-control policy to account for available resources. Each uses customized drivers to invoke APIs, which then provision the resources, *e.g.*, OpenStack nova for VM, or AL2S OSCARS for the dynamic circuits over Internet2. The operation of ORCA actors is depicted in Fig. 3, with red boxes showing the enhancements that were added to support the generalized slice-modification capability.

Reservation Modify Support. The core of ORCA consists of three types of distributed, autonomous ORCA actors that interact with each other to maintain reservation states. Each reservation passes through different states using well defined state machine actions for each actor. Reservations are associated with dictionaries of properties that describe their configurations according to the type of each reservation. Providing support for slice modification required that we first enable changes to slivers/reservations, allowing the controller logic to combine such modifications with other reservation operations. We extended the internal reservation-state machine to support new states for reservations and introduced new inter-actor calls to communicate modification actions.

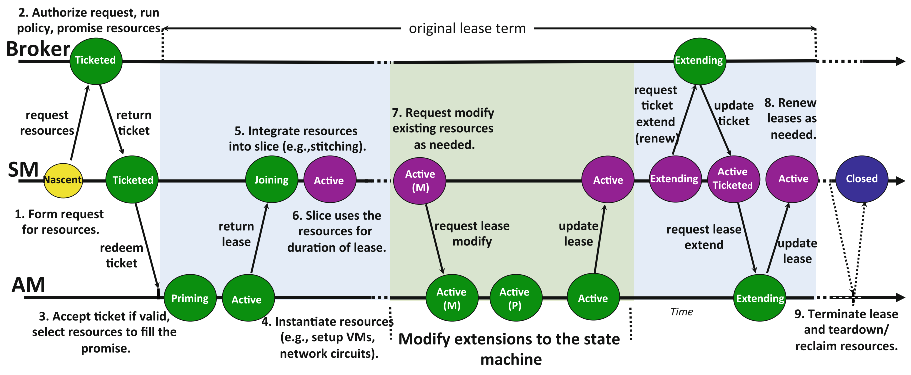
2. Authorize request, run policy, promise resources.
original lease term
Broker
Ticketed
Extending
request resources
return ticket
request ticket extend (renew)
update ticket
8. Renew leases as needed.
5. Integrate resources into slice (e.g.,stitching).
7. Request modify existing resources as needed.
SM
Nascent
Ticketed
Joining
Active
Active (M)
Active
Extending
Active Ticketed
Active
Closed
1. Form request for resources.
redeem ticket
return lease
6. Slice uses the resources for duration of lease.
request lease modify
update lease
request lease extend
update lease
AM
Priming
Active
Active (M)
Active (P)
Active
Extending
3. Accept ticket if valid, select resources to fill the promise.
4. Instantiate resources (e.g., setup VMs, network circuits).
Modify extensions to the state machine
Time
9. Terminate lease and teardown/ reclaim resources.

Fig. 4. Extended ORCA state machine.

Modifying a reservation involves updating the dictionary of properties and invoking a substrate-update task to implement those changes. For example, adding a new network interface involves adding new properties that specify, *e.g.*, its MAC and IP addresses along with the VLAN tag to which it attaches. Then, a substrate update task can be invoked, *e.g.*, on OpenStack, to equip an existing VM with a new interface that possesses the desired properties.

The new reservation states for each actor were designed to support progressive modification of stored reservation properties, and had to be reconciled with the existing state transitions. The state machine transitions on each of the three types of ORCA actors are shown in Fig. 4. The area of the figure that is shaded in green shows the extra states and transitions that were introduced to support modification of reservations. We added a new "pending" reservation state called *ModifyingLease*, shown as (M) in the figure. We also extended the ORCA management API, *i.e.*, the API used by external entities to interface with the ORCA core, to support calls from the controller for reservation modifications.

Stateful Slice Control. Conceptually, a slice is a resource container. It holds a set of resource reservations (slivers) that are granted by the providers for a specific term. We implemented the ORCA controller to maintain the current state of the slice, including individual reservations and their respective topologies. The controller also sequences the generation of new reservations, removes old reservations, and modifies existing reservations, all of which are steps in a single slice modification. Each slice-modification request consists of a subgraph that is either added or removed from the existing topology graph. Based on this slice-modification request, the controller computes sets of reservations to be added and to be removed, along with modification operations for existing affected reservations. It sequences them properly using dependency tracking and submits them to the ORCA core for execution. The standard ORCA core mechanisms communicate among the various relevant actors, making sure that resources are instantiated, deleted, or modified to reflect the new topology of the slice.

Slice-modification requests are not idempotent. They are always based on the latest slice-manifest document, reflecting the latest topology. Elements of the slice have unique URL names in the manifest. To stitch new elements of the slice these names are included in slice-modification requests as reference points to be used in the attachment of new reservations to existing ones and in the removal of existing reservations. Replaying a scale-up/out modification request results in the generation of duplicate subgraphs that are added to the slice.

The dependency-tracking mechanism builds on previously described work [2]. To support slice modifications, we extended this mechanism to include a new type of dependency. It is used in the initiation of reservation modifications only after prerequisites have been fulfilled, *e.g.*, after creating or removing a reservation. The prerequisites may provide information that is needed to invoke the reservation modification, such as the tag of a VLAN from which a node needs to be disconnected because the associated link was removed. Originally, the tracking of dependencies supported only the sequencing of new reservation events.

Persistent Resource Models. ORCA is a complex distributed system with many actors. Thus, it can experience a variety of failure modes. Recovery of state is critical; the affected portion of the system must be able to resume its state correctly and to proceed with its operation after a failure. ORCA recovery is based on two different approaches, *i.e.*, recovery of slice - or substrate topologies and recovery of reservation states in other actors such as brokers and AMs. The topology information models in ORCA are represented in Semantic Web RDF and OWL models, which are implemented using the popular RDF library Jena. Jena has a TDB component for persistent storage and query from the disk. We implemented our slice-recovery function by using TDB to save all the substrate information models at each actor and the slice models in the controller. When an actor crashes or is shut down and restarted, it can invoke the recovery function to restore its models from the latest committed state in the TDB store. The recovery function automatically reconciles the actor's information with the available resource information acquired from the ORCA core, which has its own recovery capability based on storing resource-accounting information in a relational database. Once the recovery is complete, the actor resumes its operation.

2.3 Aggregate Manager Control

ExoGENI AMs control the physical substrate and instantiate resources requested by the user via a controller. Addition and removal of network links require, respectively, addition and removal of network interfaces to existing virtual machines. This modification is handled by the AM.

Most compute aggregates in ExoGENI are individual hybrid OpenStack/xCAT clouds. ExoGENI uses a custom plugin to OpenStack networking that supports dynamic addition and removal of Layer-2 networks as well as of the corresponding network interfaces on existing virtual machines.

2.4 User Tool and API

The Flukes GUI and the NDLLIB library are the two user interfaces to ExoGENI that support the new slice-modification features. The Flukes GUI has been enhanced to support the different types of scaling behaviors described earlier.

The NDLLIB library provides a programmable interface to ExoGENI providing procedural interface to the controller API. The library consists of Java classes and methods that simplify the generation and parsing of NDL requests and manifests. The library uses an abstraction of Java objects to model slices and resources within slice topologies, *e.g.*, compute nodes, network links, and storage nodes. Users can add, remove, and modify resources in this slice model. Then, they can programmatically generate a NDL request and submit to ExoGENI controller to instantiate new resources or to change existing resources.

3 Use Cases and Evaluation

Today, the ExoGENI testbed has some twenty cloud sites across the world, the highest concentration of which is located in the U.S. Dynamic circuits must be provisioned over, *e.g.*, AL2S, the Exchange Domain, and regional networks to provision slices over multiple sites. With this in mind, we first studied the scaling performance of ExoGENI in provisioning virtual clusters scaling out to various sites. Figure 5 shows a screenshot of the user interface in Flukes from this work. The background view is the request for five sub-clusters of size 12 at different sites, expressed in the simple virtual-cluster (VC) abstraction using the *node group*. The front view is the *manifest* that shows the result, which is a slice with the instantiated VMs interconnected to the virtual broadcast link in the Exchange domain via five multi-domain dynamic paths.

Figure 6 shows the provisioning time of a VC with different numbers of nodes in three scenarios. The VC is embedded, respectively, in one cloud site, in three different sites, and in five different sites. The most important observation is that the time increases with the number of sites involved. Examination of the provisioning stages in more detail yields further insights: (1) There may be limited

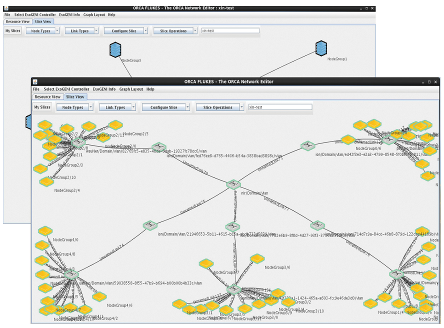

Fig. 5. VC seen in Flukes

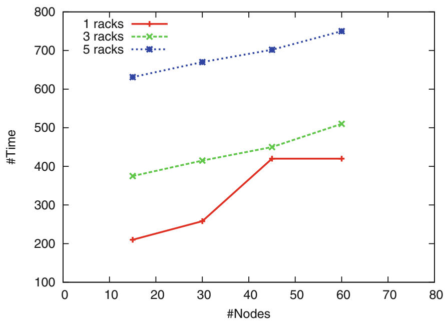

Fig. 6. VC scaling performance

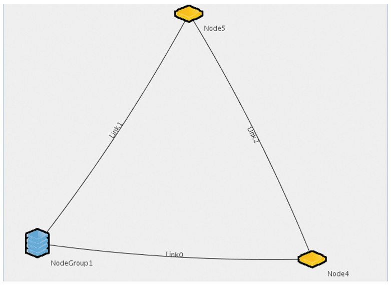

Fig. 7. Original topology

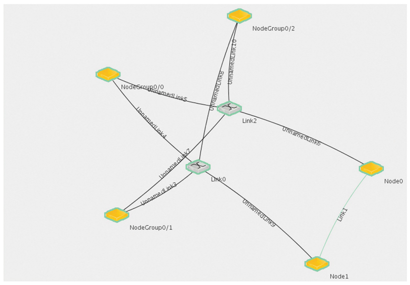

Fig. 8. Original manifest

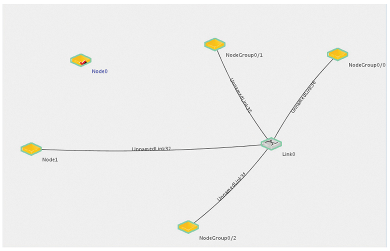

Fig. 9. Delete links

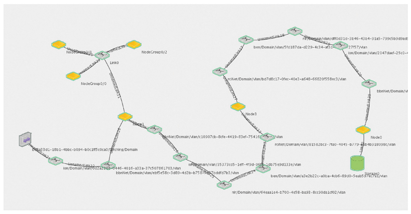

Fig. 10. Add links and nodes

resources available at a site, which affects results. For example, at the time of the experiment, if only 52 cores (standard VM resources) are available at many sites, VC size cannot be increased to 60. (2) If the entire VC is embedded in a single site, the majority of provisioning time is spent in creating the VMs. For example, when the VC size is 45, 75% of the time is used in VM creation in the OpenStack rack, and most of the rest is spent by ORCA to generate the 45 ORCA reservations. (3) In the multiple-site case, the majority of provisioning time is spent in configuring the network circuits, including the point-to-point circuits in AL2S and the multi-point circuits in the Exchange domain. Due the highly parallel configuration process in ORCA, VM creation occurs in parallel with the creation of the circuits. In the 60-node five-site case, about 35% of the time is spent on AL2S and 35% on exchange-domain configuration, whereas the remaining 30% is spent in ORCA. Circuit-creation times increase in proportion with the number of sites used, as more path computation and configuration are needed. (4) ORCA computation time also rises in proportion with the number of sites involved, as more path- and stitching computations are needed and as the reservation-management load increases.

Next, we demonstrate the modification sequence for a general topology slice with live screenshots from Flukes. The time-performance results (not shown) are dominated by the path-provisioning time. Screenshots show that a slice can be modified arbitrarily in unlimited steps, an example of the kind of capabilities inherent in the use of LEGO-like constructs to build complex structures with a limited number of building-block types.

Specifically, Fig. 7 shows the original virtual topology request. Figure 8 shows the manifest as the the slice is embedded in a single site. Next, certain links are deleted, as is shown in Fig. 9. The last one (Fig. 10) shows the result of a series of additions that include a new inter-rack path to a new node, a storage node, and a stitching port linking to a facility outside the testbed.

4 Conclusions

The present paper describes the newly developed slice-modification capabilities in the ExoGENI testbed with emphasis on the scaling of topologies. The underlying advanced control software architecture is focused on the federation of cloud and network substrates. The enhancements presented here yield a comprehensive, efficient IaaS system that gives users great flexibility in controlling and programming the scales and topologies of virtual systems.

We are developing more powerful APIs and libraries that will utilize slice-modification functions more fully, thus allowing users and applications to automate the scaling of virtual topologies.

References

1. Baldin, I., Chase, J., Xin, Y., Mandal, A., Ruth, P., Castillo, C., Orlikowski, V., Heermann, C., Mills, J.: Exogeni: a multi-domain infrastructure-as-a-service testbed. In: McGeer, R., Berman, M., Elliott, C., Ricci, R. (eds.) GENI: Prototype of the Next Internet. Springer-Verlag, New York (2016)
2. Baldine, I., Xin, Y., Mandal, A., Heermann, C., Chase, J., Marupadi, V., Yumerefendi, A., Irwin, D.: Orchestration, autonomic cloud network: a GENI perspective. In: 2nd International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010), Co-Located with GLOBECOM 2010, December 2010
3. Baldine, I., Xin, Y., Mandal, A., Ruth, P., Yumerefendi, A., Chase, J.: Exogeni: a multi-domain infrastructure-as-a-service testbed. In: TridentCom: International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, June 2012
4. Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Towards predictable datacenter networks. In: Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM 2011, pp. 242–253. ACM, New York (2011)
5. Berman, M., Chase, J.S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., Seskar, I.: GENI: a federated testbed for innovative network experiments. Comput. Netw. **61**, 5–23 (2014)
6. Chard, R., Bubendorfer, K., Ng, B.: Network health and e-science in commercial clouds. Future Gener. Comput. Syst. **56**, 595–604 (2016)
7. ExoGENI websites. http://www.exogeni.net, http://wiki.exogeni.net
8. Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A.: SHARP: an architecture for secure resource peering. In: Proceedings of the 19th ACM Symposium on Operating System Principles, October 2003

9. Gibbons, P.B.: Big data: scale down, scale up, scale out. In: Keynote Talk at the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2015), Hyderabad, India, May 2015
10. Hwang, K., Bai, X., Shi, Y., Li, M., Chen, W.-G., Wu, Y.: Cloud performance modeling with benchmark evaluation of elastic scaling strategies. IEEE Trans. Parallel Distrib. Syst. **27**(1), 130–143 (2016)
11. Irwin, D., Chase, J.S., Grit, L., Yumerefendi, A., Becker, D., Yocum, K.G.: Sharing networked resources with brokered leases. In: Proceedings of the USENIX Technical Conference, June 2006
12. Kang, J.-M., Lin, T., Bannazadeh, H., Leon-Garcia, A.: Software-defined infrastructure and the SAVI testbed. In: Leung, C.V., Chen, M., Wan, J., Zhang, Y. (eds.) Testbeds and Research Infrastructure: Development of Networks and Communities: 9th International ICST Conference, pp. 3–13. Springer International Publishing, Cham (2014)
13. McGeer, R., Berman, M., Elliott, C., Ricci, R. (eds.): GENI: Prototype of the Next Internet. Springer-Verlag, New York (2016). In production for publication, July 2016
14. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2Nd ACM Symposium on Cloud Computing, SOCC 2011, pp. 5:1–5:14. ACM, New York (2011)
15. Xin, Y., Baldin, I., Heerman, C., Mandal, A., Ruth, P.: Scaling up applications over distributed clouds with dynamic layer-2 exchange and broadcast service. In: ITC'26 Workshop on Federated Future Internet and Distributed Cloud Testbeds (FIDC 2014), Karlskrona, Sweden, September 2014