GENICloud and TransCloud: Towards a Standard Interface for Cloud Federates

Andy Bavier
Princeton University
acb@cs.princeton.edu

Yvonne Coady
University of Victoria
ycoady@cs.uvic.ca

Tony Mack
Princeton University
tony.mack@gmail.com

Chris Matthews
University of Victoria
cmatthew@cs.uvic.ca

Joe Mambretti
Northwestern University
j-mambretti@northwestern.edu

Rick McGeer
Hewlett-Packard Laboratories
rick.mcgeer@hp.com

Paul Mueller
TU-Kaiserslautern
pmueller@informatik.uni-kl.de

Alex Snoeren UCSD
snoeren@csucsd.edu

Marco Yuen
Princeton University
marcoy@gmail.com

ABSTRACT

In this paper, we argue that federation of cloud systems requires a standard API for users to create, manage, and destroy virtual objects, and a standard naming scheme for virtual objects. We introduce an existing API for this purpose, the Slice-Based Federation Architecture, and demonstrate that it can be implemented on a number of existing cloud management systems. We introduce a simple naming scheme for virtual objects, and discuss its implementation.

1. Introduction

"Cloud" systems and services refer to the remote allocation and use of various virtual resources over the Internet. These resources can range from virtual machines, block stores, and databases (Amazon EC2 and S3, Rackspace, HP Cloud), to threads (Microsoft Azure) to language engines and VMs (Google App Engine). OpenCirrus[1, 20] offers a federated collection of local clusters, each of which offer a number of virtual resources. The US Global Environment for Network Innovations[12] project offers a heterogeneous collection of virtual networked resources for researchers in networking, distributed, and cloud systems.

One model for federated systems, the so-called "Cloud-bursting" model, involves the use of a relatively small number of homogeneous systems. In this model, a secondary cloud system is used for reasons of capacity or cost when the primary cloud system is saturated. A second model, which is of interest to us, is when a large number of potentially heterogeneous resources are allocated from a large number of different systems, from different administrative domains. There are a variety of use cases which motivate this second, "Ubiquitous Cloud" model. Content distribution systems which require proximity to widely-distributed users; robust stores for critical data, dispersed to prevent damage or destruction in the event of local catastrophes; wide-area sensing systems with heavy-computation based back ends; experimentation on wide area distributed systems or cloud infrastructure systems; and so on.

An immediate and compelling need for the Ubiquitous Cloud model is the need to create large-scale testbeds for the research and education community. Industrial practice has moved far beyond the capacity of academic research. A Yahoo! "clique" of servers, an internal unit of management, comprises 20,000 servers and perhaps a quarter-million cores. No single academic testbed has anything like that capacity.

The Ubiquitous Cloud model exposes a number of issues in scalability of cloud systems. The most prominent of these is the need for a common API across heterogeneous systems. Large-scale distributed systems will typically not be instantiated or managed by a Graphical User Interface, but by tools. Indeed, users today largely instantiate Cloud jobs on Eucalyptus[9], OpenStack[21] or some commercial systems using the popular command-line euca2ools[8]. The euca2ools have hard-coded the API to Eucalyptus systems, which was designed to be API-compatible with the popular commercial offering Amazon EC2; OpenStack, a later, open-source entrant, supports this API.

Though the Amazon API is emerging as a de facto standard in this area, it has a number of weaknesses as a general solution. The primary weakness is that it is tuned specifically for virtual machine based compute resources and block stores, and so it has spread only across systems that offer roughly the same set of services that Amazon does. Allocation of resources across heterogeneous systems (such as, for example, allocating a collection of Restricted Python[7] processes from the Seattle[6, 18, 30] testbed, a collection of bare hardware nodes from ProtoGENI[28], some cluster-based compute nodes from GENI-Cloud[16], and some wireless or wimax nodes will require an API that permits the user to specify, and the cloud systems to advertise, not simply the number but the *kind* of available resources.

Authorization and authentication present significant challenges. Different organizations control different facilities, and operate in different economic and legal environments. It is impractical to expect that these organizations will concur on a common user database, common set of roles, common passwords, or common Acceptable Use Policies. Federated authentication solutions such as Shibboleth[32] have been used, but

present some logistical difficulties: in particular, Shibboleth assumes a persistent and reliable connection between the organization providing the facility and the organization certifying the user, and a single oracle for any particular user. Again, this has been shown to work for a collection of relatively homogenous environments, but presents obvious problems when used for a heterogeneous set of facilities.

In this paper, we discuss the use of the GENI standard Slice-Based Federation Architecture[25] as a standard API for heterogeneous cloud systems. The SFA incorporates a simple API which fits well on existing cloud systems, and augments this with a resource-specification standard for both requests and allocation, the RSpec. It further incorporates an authorization scheme based on unforgeable, self-validating certificates, eliminating the need for a centralized database, clearinghouse, or federated identity mechanism. We show that it has been used succesfully on a wide variety of virtual resource-as-a-service facilities.

Our contribution in this paper is to describe an implementation strategy for the SFA onto legacy Cloud systems. We have successfully used an SFA implementation to control both OpenStack and Eucalyptus-based clusters, and use it today to offer seamless federation between a PlanetLab-based infrastructure and an OpenStack-based infrastructure.

Further, when a user creates a large collection of virtual resources, he must be able to name and access them predictably and easily without manually naming each one. We propose a simple URL-based naming scheme for these resources.

The remainder of this paper is organized as follows. In Section 2, we give a high-level overview of the SFA, with some short examples of its use. In Section 3, we describe a simple implementation strategy for the SFA on legacy cloud systems that we have used succesfully on OpenStack and Eucalyptus. In Section 4 we describe TransCloud, a naming scheme for large, heterogeneous, multi-site slices of clouds. In Section 5 we draw some conclusions, and offer suggestions for future work.

2. The Slice-Based Federation Architecture

The Global Environment for Network Innovations (GENI) program is a US National Science Foundation program to construct a next-generation testbed for networking and distributed systems research. It was designed by a team of over 50 US systems researchers in 2006-2007, following the success of two precursor systems: Emulab[10, 35] a hardware-as-a-service cluster at the University of Utah devoted to network experiments and emulation, and PlanetLab[4, 24, 27, 33], a worldwide containers-as-a-service distributed systems platform developed by Princeton University, UC-Berkeley, the University of Washington, Intel, and HP.

GENI soon grew to encompass a number of other control frameworks and testbeds:

1. ORBIT[22], an experimental testbed for wireless nodes from the University of Rutgers

2. Seattle[30], a high-level distributed Python environment from the University of Washington and NYU-Poly

3. FOAM[11], a programmable-networks-as-a-service system from Stanford, UC-Berkeley and BigSwitch systems, which offers isolated layer-2 networks based on OpenFlow[17, 19] and FlowVisor[31] technology.

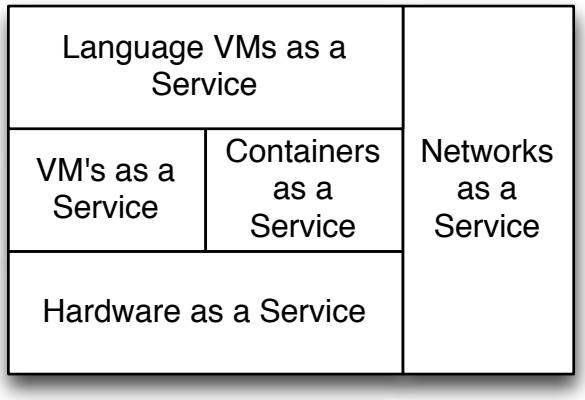

Figure 1: A Simplified Form of the As A Service Stack

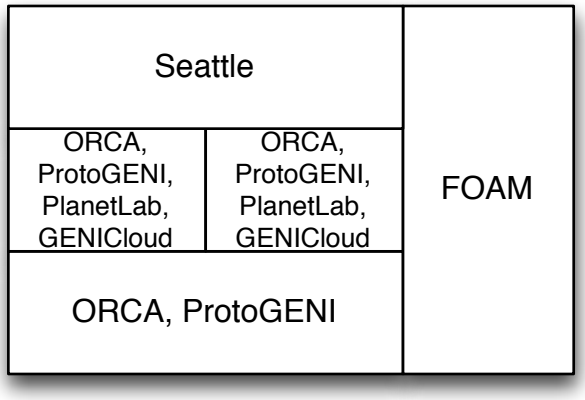

Figure 2: A Simplified Form of the GENI Stack

4. ORCA[2, 3], a substrate management system from Duke University offering multiple physical and virtual compute resources.

A simplified version of the various computing elements offered as services is shown in Figure 1. Most of these elements are available from commercial providers. Google App Engine, for example, is an excellent example of language VM's as a stack.

The stack is both more complex than is shown in Figure 1 and contains more elements; we have simplified it for explanatory purposes. For example, Language VM's can rest directly on hardware, and need not be mediated through virtual machines or containers, and containers can rest on VM's if desired. Further, Storage as a Service (such as Amazon S3) is another vertical pillar. Nonetheless, this serves to illustrate the point: distributed systems can be built from a combination of virtual resources of various types, and an API must accomodate those.

The GENI frameworks fit into the services stack in Figure 1, with the mapping shown in Figure 2. Again, this is a simplified picture; it omits ORBIT, and has the same simplifications as in Figure 1 .

GENI experiments could be expected to span any subset of these control frameworks. Moreover, each of these was a codebase, which could be expected to be replicated over a number of administrative domains. Large experiments are expected to require the marshalling and orchestration of large numbers of

heterogeneous resources. In sum, software tools were expected to create and manage disparate resources spread over multiple administrative domains, running under different management suites. A common API spanning all of these control frameworks was required, with a common authorization method and a common method of advertising resources and servicing resource requests. The common API is the *Slice-based Federation Architecture*. We detail it here.

The central abstraction in the Slice-Based Federation Architecture is a *slice* of the underlying physical substrate. A slice refers to the entire collection of virtual compute, communication, and storage resources devoted to an experiment. One can think of it as a virtual, distributed network of virtual machines devoted to a specific project. For example, the *CoDeeN*[23, 34] slice on PlanetLab consists of several hundred Linux VServers[15], distributed at several hundred sites worldwide, which together implement a Content Distribution Network. The GENIS3Monitor slice[5] is a scalable, extensible, and safe network monitoring system for research networks and clouds.

The notion of a slice is not common among Cloud management frameworks, but it provides a number of services to end-hosts and users. For example, it makes possible global manipulation of the elements of a slice as a unit, without separate operations on each individual element. Good examples of this are loading user certificates onto every individual element of the slice, or doing software loads and boot scripts on every element of the slice. It also provides a natural unit of resource allocation and accounting.

Individual virtual resources within slices are referred to as *slivers*. Again, a sliver is simply a generalization of the concept of a virtual machine to encompass containers, physical machines, or even individual named block stores.

The API exported by the SFA is similar to that exported under the euca2ools, or the OpenStack API, with an additional overlaying set of APIs to manage sets of resources, or slices. It is an XML/RPC interface over HTTPS, with authentication by credential.

The central data structure in the SFA is the Resource Specification, or RSpec[29]. It is GENI's mechanism for advertising, requesting, and describing the resources used by a slice. In general, it is an XML document. For a machine it will often describe the instance size requested, whether it needs exclusive access to a physical host (i.e., requires a host or can live with a VM), may describe a particular image to be loaded, and so on. We show one simple example abstracted from the ProtoGENI RSpec examples, with various namespace details omitted for space in clarity in Figure 3. The full example can be found at: `http://www.protogeni.net/trac/protogeni/wiki/RSpecRequestDiskImageExample`

The RSpec example shown in Figure 3 requests a physical host ("raw-pc") and a Fedora Core 10 OS. This is an example of a *request* RSpec, which would be used by a tool to request a physical node with a specific OS. In addition, an *advertisement* RSpec is used by a manager of physical resources to show what is available. A simplified example is shown in Figure 4, taken from `http://www.protogeni.net/trac/protogeni/wiki/RSpecAdSingleNodeExample`

This particular RSpec describes a node with name "pc160", which is of type "pc850" (850 MHz x86 processor), currently available for exclusive use. It also matches requests for "pc" (any x86-based processor). It has two network interfaces.

```
<rspec type="request">
  <node client_id="exclusive-0"
      component_manager_id="
      urn:publicid:IDN+emulab.net+
      authority+cm" exclusive="true">
    <sliver_type name="raw-pc">
      <disk_image name="urn:publicid:IDN+
          emulab.net+image+emulab-ops//
          FEDORA10-STD"/>
    </sliver_type>
  </node>
</rspec>
```

Figure 3: ProtoGENI Request RSpec Example

```
<rspec    type="advertisement"
    generated="2009-07-21T19:19:06Z"
    valid_until="2009-07-21T19:19:06Z" >
  <node component_manager_uuid="
      urn:publicid:IDN+emulab.geni.emulab.
      net+authority+cm"
        component_name="pc160"
        component_uuid="urn:publicid:IDN+
            emulab.geni.emulab.net+node+
            pc160" >
    <node_type type_name="pc850"
        type_slots="1" />
    <node_type type_name="pc" type_slots=
        "1" />
    <available>true</available>
    <exclusive>true</exclusive>
    <interface component_id="
        urn:publicid:IDN+emulab.geni.
        emulab.net+interface+pc160:eth0"/>
    <interface component_id="
        urn:publicid:IDN+emulab.geni.
        emulab.net+interface+pc160:eth1"/>
  </node>
</rspec>
```

Figure 4: ProtoGENI Advertisement RSpec Example

In addition (for reasons of space we will omit an example) there is a third type of RSpec: the Manifest Rspec. This describes a resource requested by and granted to a slice; it shows a fulfilled request. Though the example RSpec's we have shown have come from ProtoGENI, the essential features are uniform across the various GENI Control Frameworks.

Physical devices in the SFA are referred to as *Components*, which are formally defined as collections of physical resources. A simple example is a server, but others include programmable switch, Open WRT access point, dedicated software router, and so on. Components are controlled by *Component Managers* (note that in Figure 4, the node rspec had a component_manager attribute; that had the URN of manager of that component). One can think of a Component Manager as having roughly the same capabilities and duties as a Eucalyptus Node Controller.

Components in the SFA are grouped into *Aggregates*. Again, a simple example of an Aggregate is a cluster; another exam-

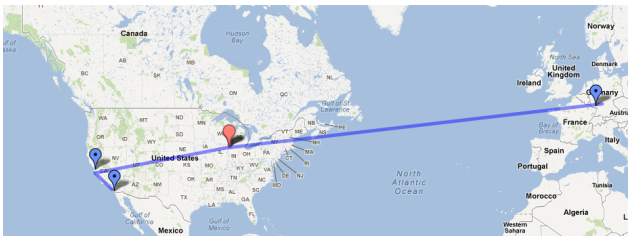

Figure 5: GENICloud Sites

ple is a distributed platform such as PlanetLab, or a collection of OpenFlow switches. An Aggregate is managed by an *Aggregate Manager*, which has roughly the same function as a Eucalyptus Cluster Controller. However, Aggregates can be hierarchical, which means that Aggregates can contain other Aggregates as well as Components. One use case for this is where a cluster such as ProtoGENI contains a set of OpenFlow switches with a distinct manager; in order to allocate a slice containing dedicated nodes hooked up to a progammable network slice, the cluster AM makes requests of the AM responsible for the collection of OpenFlow switches. This implies that the AM interface is a superset of the CM interface, since the AM must respond to component requests.

The SFA defines global identifiers (GID) for all entities in the federated system: principals, slices, physical components, services, etc.[25] The GID is an unforgeable certificate, signed by one or more entities, that binds together three pieces of information: the object's public key, a unique identifier (e.g., a UUID), and a lifetime. Any entity may verify a GID via cryptographic keys that lead back, through a chain of endorsements, to a well-known root; in this way the receiver of a GID can authenticate that the sending object is the one to which the GID was actually issued. A GID signed with its own private key is called a self-certifiying ID or SCID. A SCID establishes the issuing entity's right to assert attributes or authorization rules for the named object.

3. Implementing the SFA on Legacy Cloud Management Systems

The SFA has proven to be an effective abstraction over existing legacy infrastructures such as Emulab and PlanetLab, and thus a candidate as an effective overlying API for general as-a-Service computational infrastructures. Further, it has been able to accomodate unanticipated infrastructures such as virtualizable switching fabrics (OpenFlow).

We have been investigating whether the SFA can be used as an overlying API for general cloud frameworks in the GENICloud[16] project. We sought to answer three questions:

1. Can the SFA overlay an existing cloud management system, designed entirely independently of the SFA?
2. Can we manage clusters using *different* specific management systems, each of which presents an SFA interface, and create slices which span the different systems?
3. Can we use the SFA to manage a multi-site cloud system?

We built a four-site Cloud system, using varying underlying resources to manage each site, and managing them with the SFA. We show the sites in Figure 5. The four are HP Labs, as part of the OpenCirrus cluster[20], Northwestern University, UC San Diego, and the Technical University of Kaiserslautern.

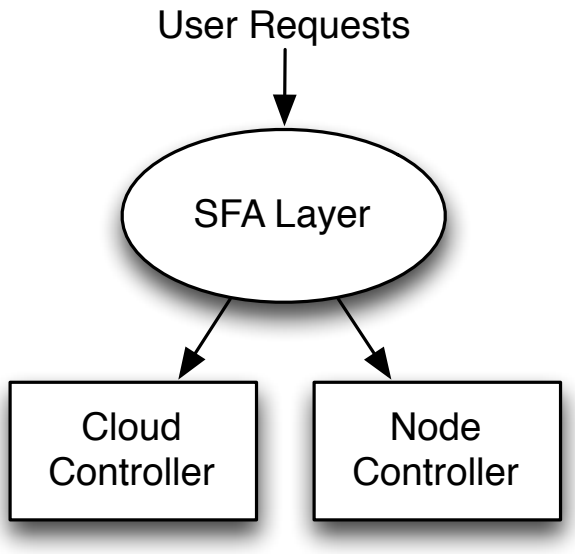

Figure 6: SFA-on-Cloud Architecture

The four sites run different management stacks on the local clusters, but appear homogeneous through the SFA.

A further motivation for the GENICloud effort was to federate cloud systems into the distributed and experimental infrastructures provided by GENI. Current GENI platforms largely are not designed to scale rapidly on demand. Under the federation of a standard Cloud platform and GENI, a more comprehensive platform is available to users; for example, development, computation, data generation can be done within the cloud, and deployment of the applications and services can be done on distributed platforms (e.g., PlanetLab). By taking advantage of cloud computing, GENI users can dynamically scale their services on GENI depending on the demand of their services, and benefit from other services and uses of the cloud. Take a service that analyzes traffic data as an example; the service can deploy traffic collectors to collect Internet traffic data on PlanetLab . The traffic collected can be stored and processed in the cloud.

PlanetLab, being a part of the GENI project as one of the control frameworks, has high global penetration. However, it was not designed for either scalable computation nor large data services. Some services and experiments require a huge amount of data or they need to persist a large amount of instrumental data; also, it lacks the computation power for CPU intensive services or experiments. GENICloud fills in the gap by federating heterogeneous resources, in this case, a cloud platform with PlanetLab.

Most of the implementation effort of GENICloud concentrated on implementing the aggregate manager on top of the existing Cloud Controller, under both Eucalyptus and OpenStack. A key design choice was whether to modify the existing Controller, or use an overlying controller which implemented SFA API calls by calling through to the underlying controller.

We chose the latter, and the resulting architecture is shown in Figure 6. In this architecture, both the fact that the user is coming through an SFA layer and the identity of the underlying controller is hidden. The SFA aggregate manager acts an mediator between user requests and an underlying cloud. The primary advantage is that we need not maintain modifications

and patches in existing Cloud managers, updating them as new modifications come out; rather, we only need up date our Aggregate Manager as the interface to the Cloud manager changes. We have not yet implemented, but do not exclude, Cloud controller specific optimizations for controllers with specific optimizations such as Tashi[13]. A secondary advantage is that the identity of the underlying Cloud controller is hidden from the user, meaning that user-facing tools and scripts don't change when the underlying Cloud controller is hidden.

The Aggregate Manager manages the creation of cloud instances for a slice, and maintains a mapping of slices and instances. The Aggregate Manager offers RSpecs for the VM's and containers created by the underlying cloud, and manipulates the underlying Node Controller and Cluster Controller to serbvice user requests.

The resource specification format for VM's and containers closely mirrors the RSpec formats seen previously, and indeed we were able to re-use the ProtoGENI RSpecs for Virtual machines, and the PlanetLab RSpecs for containers.

In our design, the SFA is essentially a proxy between the user and OpenStack. Users authenticate to the SFA and are authorized to perform specific actions in the SFA API using SFA credentials. All authorization policy checks occur in the SFA layer. This insulates the Cloud Controller from any overlying policy changes at the SFA layer, such as the proposed move towards Attribute-Based Access Control, or ABAC, a self-verifying capability system[14].

We have implemented the SFA layer over OpenStack and Eucalyptus. Currently, our Palo Alto site is running OpenStack under the SFA, and UCSD, Northwestern and TU-Kaiserslautern are running PlanetLab under the SFA, using the MyPLC cluster controller[26]. We have succesfully transitioned the Palo Alto site from Eucalyptus to MyPLC to OpenStack, without changing the overlying SFA interface.

4. Naming Scheme: TransCloud

A key for large-scale, multi-site, multi-administrative domain slice is the ability for the user to administer his slice, orchestrate the action of the slice's slivers, and easily configure the slivers in his slice to communicate. In order to do this, a standard naming scheme for slivers is required. DNS services will take care of communication when the slice is set up.

Here, we propose such a naming scheme and DNS implementation. Our scheme is inspired by the naming scheme of Emulab[10]. In Emulab, the fundamental subdomain is a "project", which is a group of researchers who create "experiments" in pursuit of some research goal. Emulab's experiment is simply a slice; a project is an overlying space used for resource allocation, directory structures, storage of related data, and also provides a namespace to permit researchers to isolate namespaces.

Using the terminology that we've been using in this paper, each sliver in an Emulab slice is named `<sliver-name>.<slice-name>.<project-name>.emulab.net`; e.g., `sender.baselineqos19.chart.emulab.net`. In this case, `sender` names the node, `baselineqos19` the experiment, and `chart` the project. Nodes within a slice can simply use the `<sliver-name>` and this resolves to the appropriate FQDN. Use of project as a namespace partition is not in the SFA and is not a feature of other control frameworks such as PlanetLab, but it has its uses. In particular, PlanetLab prepends the name of the host institution on a slice name to prevent slice namespace contention; the CoDeen slice is not`codeen`, but rather `princeton_codeen`. The PlanetLab convention is an alternative.

Emulab achieves this by using its own internal DNS servers and updating the DNS entries when an experiment is swapped in. We propose to use the same method.

Emulab was conceived as a single-site, single-administrative domain testbed. We are designing for multi-site, multi-domain administration, so we modify the Emulab scheme to accomodate this. This means that we must incorporate both the site name and the administrative domain name in the URL; both must be accomodated because the *site* of a sliver can be important (consider a Content Distribution Network or real-time sensitive server, for example); and the administrative domain is the entity that will actually allocate the slice.

There are two fundamental principles that underlie our scheme. First, administrative domains must make only three agreements: they will run the SFA, accept GID's as credentials, and implement the TransCloud naming schement. The second principle is that names should be allocated to slices and projects at highest level in the hierarchy possible.

The first principle argues that projects and slices do not span administrative domains, since this would imply that administrative domains need to agree on a common project namespace. The second argues that slices and projects should be defined at the administrative domain level, since a single administrative domain can administer projects and slices that span multiple sites, and there is value to users in creating multi-site slices.

We have obtained `trans-cloud.net` as the root domain, so our scheme is then: `<sliver-name>.<site-name>.<slice-name>.<project-name>.<domain-name>.trans-cloud.net`. Again, appropriate DNS work will permit the use of `<sliver-name>.<site-name>` within a slice.

Root storage at a site, and global ssh logins, should be to `users.<site-name>.<domain-name>.trans-cloud.net`; this is the equivalent of `users.emulab.net`.

Interactions with the Domain authority occurs through `<authority-name>.trans-cloud.org`; the `.org` domain is for administrative access (e.g., certificate upload and renewal, slice creation and management, and the like); the `.net` domain for experimental access.

5. Conclusions

In this paper we have demonstrated the implementation of SFA on two legacy Cloud architectures, and shown the feasibility of a cross-site, cross-manager SFA implementation. GENI-Cloud is a single administrative domain spanning four sites and two aggregate managers. We have proposed a naming scheme for a unified, multi-site, multi-domain distributed cloud infrastructure.

Acknowledgements

This work was partially supported by the GENI Project Office under contract GENI 1779B. The GENI Project Office is funded by the National Science Foundation's CISE Division.

6. References

[1] A. Avetisyan, R. Campbell, I. Gupta, M. Heath, S. Ko, G. Ganger, M. Kozuch, D. O'Hallaron, M. Kunze, T. Kwan, K. Lai, M. Lyons, D. Milojicic, H. Y. Lee, Y. C.

Soh, N. K. Ming, J.-Y. Luke, and H. Namgoong. Open cirrus: A global cloud computing testbed. *Computer*, 43(4):35 –43, april 2010.

[2] I. Baldine, Y. Xin, A. M, C. Heermann, J. Chase, V. Marupadi, A. Yumerefendi, and D. Irwin. Networked cloud orchestration: A geni perspective. In *2010 Globecom Workshops*, 2010.

[3] I. Baldine, Y. Xin, A. Mandal, P. Ruth, A. Yumerefendi, and J. Chase. Exogeni: A multi-domain infrastructure-as-a-service testbed. In *Proceedings Tridentcom, 2012*, 2012.

[4] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI*, May 2004.

[5] E. Blanton, S. Chatterjee, S. Gangam, S. Kala, D. Sharma, S. Fahmy, and P. Sharma. Design and evaluation of the s^3 monitor network measurement service on geni. In *COMSNETS*, pages 1–10, 2012.

[6] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. *SIGCSE Bull.*, 41(1):111–115, 2009.

[7] J. Cappos, A. Dadgar, J. Rasley, J. Samuel, I. Beschastnikh, C. Barsan, A. Krishnamurthy, and T. Anderson. Retaining Sandbox Containment Despite Bugs in Privileged Memory-Safe Code. In *The 17th ACM Conference on Computer and Communications Security (CCS '10)*. ACM, 2010.

[8] Euca2ools. `http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3`.

[9] Eucalyptus. `http://www.eucalyptus.com/`.

[10] Emulab website. `http://www.emulab.net`.

[11] Foam. `https://openflow.stanford.edu/display/FOAM/Home`.

[12] Geni project website. `http://www.geni.net`.

[13] M. A. Kozuch, M. P. Ryan, R. Gass, S. W. Schlosser, D. O'Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G. R. Ganger. Tashi: location-aware cluster management. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACDC '09, pages 43–48, New York, NY, USA, 2009. ACM.

[14] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[15] Linux vservers. `http://linux-vserver.org/Welcome_to_Linux-VServer.org`.

[16] R. McGeer, A. AuYoung, A. Bavier, J. Blaine, Y. Coady, J. Mambretti, C. Matthews, C. Pearson, A. Snoeren, and M. Yuen. Transcloud: Design considerations for a high-performance cloud architecture across multiple administrative domains. In *Proceedings CLOSER*, 2011.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev*, 38:69–74, March.

[18] Monzur Muhammad and Justin Cappos. Towards a Representive Testbed: Harnessing Volunteers for Networks Research. In *The First GENI Research and Educational Workshop*, GENI'12, 2012.

[19] Openflow website. `http://www.openflow.org`.

[20] Opencirrus. `http://opencirrus.org/`.

[21] Openstack. `http://openstack.org/`.

[22] Orbit. `http://www.winlab.rutgers.edu/docs/focus/ORBIT.html`.

[23] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the web: An open proxy's view. In *HotNets*, 2003.

[24] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir. Experiences implementing planetlab. In *OSDI*, November 2006.

[25] L. Peterson et al. Slice-based federation architecture, v 2.0. `http://groups.geni.net/geni/attachment/wiki/SliceFedArch/SFA2.0.pdf`.

[26] MyPLC User guide. `http://www.planet-lab.org/doc/myplc`.

[27] Planetlab. `http://www.planet-lab.net`.

[28] Protogeni web site. `http://www.protogeni.net/trac/protogeni`.

[29] Protogeni rspec. `http://www.protogeni.net/trac/protogeni/wiki/RSpec`.

[30] Seattle. `https://seattle.cs.washington.edu/`.

[31] R. Sherwood. Safely using your production network as a testbed. *Login; Magazine*, 36(1), February 2011.

[32] Shibboleth. `http://shibboleth.net`.

[33] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using planetlab for network research: myths, realities, and best practices. In *IN PROCEEDINGS OF THE SECOND USENIX WORKSHOP ON REAL, LARGE DISTRIBUTED SYSTEMS (WORLDS)*, pages 17–24, 2006.

[34] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the codeen content distribution network. In *In USENIX Annual Technical Conference, General Track (2004*, pages 171–184, 2004.

[35] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and network. In *Proceedings of OSDI 02*, 2002.