Using OpenStack for an Open Cloud eXchange(OCX)

P. Desnoyers* J. Hennessey†, B. Holden‡, O. Krieger†, L. Rudolph§, A. Young‡,
*Northeastern University. pjd@ccs.neu.edu
†Boston University. {henn,okrieg}@bu.edu
‡Red Hat. {bholden,adam}@redhat.com
§Two Sigma. larry.rudolph@twosigma.com

***Abstract*—We are developing a new public cloud, the *Massachusetts Open Cloud* (MOC) based on the model of an Open Cloud eXchange (OCX). We discuss in this paper the vision of an OCX and how we intend to realize it using the OpenStack open-source cloud platform in the MOC. A limited form of an OCX can be achieved today by layering new services on top of OpenStack. We have performed an analysis of OpenStack to determine the changes needed in order to fully realize the OCX model. We describe these proposed changes, which although significant and requiring broad community involvement will provide functionality of value to both existing single-provider clouds as well as future multi-provider ones.**

I. Introduction

An *Open Cloud Exchange* (OCX) [3] is a model for multiple, mutually distrustfull, service providers to offer and deliver services to clients in the same cloud. It forms the foundation for a fully open and fair cloud marketplace; something not possible in today's vertically integrated clouds stood up by a single provider (e.g. AWS, Google, Microsoft Azure).

Each provider in an OCX may offer distinct services (*e.g.* compute or storage), services that build on other services (*e.g.* PaaS on top of IaaS), or may compete by providing different offerings of the same type of service (*e.g.* one compute service from IBM and another from Dell). The clients that wish to make use of these services can pick and choose between a diversity of offerings rather than being locked into the choices made by a single service provider.

The novel requirements of an OCX over a single provider cloud include: (i) exposing to consumers a marketplace with multiple providers, (ii) allowing clients to programatically choose between competing offerings based on performance, price, or other properties, (iii) allowing consumers to securely mix and match services from different providers, (iv) allowing each provider to control authentication and authorization for their service, (vi) enabling networks that span multiple providers and (vi) enabling a marketplace that allows providers at many different levels, including the hardware, to innovate and expose their value to client.

We next describe some background on OpenStack and cloud federation. We then discuss the OCX model and the *Massachusetts Open Cloud* (MOC), an example of an OCX being developed. The core of the paper discusses the how the requirements of an OCX can be supported.

II. Background

While the implications of the OCX model are profound, in many ways it is a natural progression and can be developed by modifying and extending ideas, techniques and middleware of existing commercial and academic single-provider cloud systems.

A. Cloud Software and OpenStack

A modern cloud is a complex distributed system with many diverse hardware and software components, a huge number of parameters and complex configuration settings that must all be monitored and maintained in the presence of failures. To make this practical, scalable clouds are developed using a Service Oriented Architecture (SOA) [19], where each service provides well-defined functionality, exposes its own API, and can be maintained, scaled, and evolved independently.

A good example of this architecture is the OpenStack [17] cloud middleware. Services supported by OpenStack include: Neutron for networking services, Glance for image services, Cinder for disk volumes, Nova for compute, and Keystone for authentication and authorization. Each service is itself a distributed system, but has a single well defined API that is used by both other services and by customers of the cloud.

An SOA is a necessary prerequisite of the OCX model, and our implementation of an OCX is based on modifying and extending OpenStack. Moreover, OpenStack is a good starting point due to its rich open source community, and the plugin architecture where uniform APIs support a wide variety of underlying equipment and mechanisms. As an example, the networking component has 17 different plugins for different networking infrastructure technologies.

With OpenStack, for each service, a single *API endpoint* (identified by a URI) acts as a broker and scheduler to distribute requests to the multiple underlying components of the service. In the case of a diversity of implementation, constraints provided by the user may help determine the implementation to use.

B. Cloud Federation

Research [5, 7, 15] and industry [16, 18] cloud federation projects allow a client to take advantage of multiple cloud platforms, either to augment private clouds with as-needed capacity from a public provider, or to allow clients to concurrently exploit multiple public clouds.

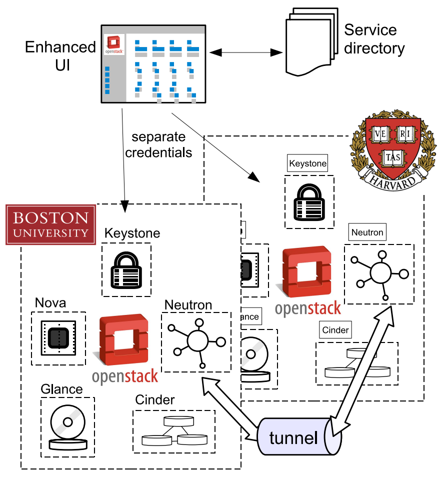

Fig. 1: Simple federation of administratively distinct clouds.

Figure 1 is an example architecture of a simple federated cloud. In it, an enhanced user interface makes use of a service directory to locate providers (in this case, Harvard and BU computing services) and their capabilities. Requests are then made directly to each provider, using credentials supplied by that provider; these requests allocate resources entirely located within that provider's domain. The user deploys network tunneling or VPN technologies to "stitch" together networks from the separate domains across the public (or data center-internal) Internet [4].

Cloud (or Grid [10]) federation can be seen as a limited form of OCX. However, existing federation realizations greatly limits the ability of a user to "mix and match" service components from different providers, *e.g.*they cannot use compute services from one provider and block storage from another. Within each cloud, the client is limited to the offerings provided by the single provider. Other limitations in this approach include (a) the need for separate credentials across a potentially large number of providers (see Section V-E), and limitations on network efficiency due to a lack of infrastructure support for bridging between providers, (see Section V-F).

III. The OCX model

In the OCX model a single trusted entity (In our case the MOC, described in more detail below) provides a small number of shared services including: a global service directory for advertising services, a service for authenticating and authorizing users of the OCX, and billing. All other services may be stood up by different providers, and a client can pick and choose between services of different providers exposed through the exchange. For example, at the IaaS layer, a client may select compute from Dell and volume storage from EMC for one virtual machine, while selecting both compute and storage from Cisco for another, and have those virtual machines communicate with each other over networks that span the different providers.

The OCX model can be viewed as a natural outgrowth of the SOA of today's cloud software, where different providers may stand up different services, and it is the client that selects which services are used. It can also be viewed as an outgrowth of cloud federation, where the federation is finer grained and at the basis of individual services rather than entire clouds.

Just as in today's single-provider clouds, we expect most clients to interact with the cloud through higher level intermediaries (e.g., Heroku, Cloud Foundry, Rightscale, Engine Yard, OpenShift, Hadoop, Spark, StarCluster). There are however two key differences. First, in an OCX an intermediary can be a first class participant in the cloud, advertised through the exchange in the same fashion as core Infrastructure as a Service (IaaS) services and fully integrated into the user interfaces, APIs, and billing model. This is in contrast to existing single provider clouds, where there is a fundamental difference between the services of the provider and the marketplace of higher-level third party services/intermediaries.

Second, the task of selecting the right services from among many potential offerings in an OCX may be difficult for consumers[1], making it important to have intermediaries that provide clients with a simple model that meets their needs. For example, an intermediary could implement an IaaS service as a broker on top of multiple underlying IaaS providers. As another example, a Spark-based [21] intermediary for big-data could exploit domain-specific information to better manage the use of IaaS services.

The exchange model also enables a richer diversity of lower level services. A service with a different interface can be exposed through the exchange, and be made available to any client that knows how to use it. For example, platforms such as GENI [11] or large HPC clusters can be exposed through the exchange and the client can mix and match between resources in these services. In fact, the Hardware as a Service (HaaS) described in Section V-H is exposed through the OCX exchange in the same way as OpenStack services, allowing different OpenStack services and clients to compete to use the hardware directly.

IV. The MOC realization

Our motivation for the Open Cloud Exchange model is as a basis for the Massachusetts Open Cloud (MOC), a new, non-profit Infrastructure as a Service (IaaS) public cloud being created as a collaboration between the Commonwealth of Massachusetts, research universities and industry. Major university partners include Boston, Harvard, Northeastern Universities, MIT and the University of Massachusetts. Industry partners include Cisco, EMC, SGI, Red Hat, Juniper, Canonical, Dell, Intel, Mellanox, Brocade, DataDirect Networks, Mathworks, Plexxi, Cambridge Computer Services, Enterprise DB and Riverbed; they are contributing engineering and operational talent, equipment, financial support and business guidance.

The goals of the MOC are to:

- Provide a computational service for both university and industry.
- Support the acquisition of specialized resources that no one institution could justify acquiring.

[1]Arguably, an advantage of today's inflexible and closed public cloud is that it offers a few easily-understood offerings.

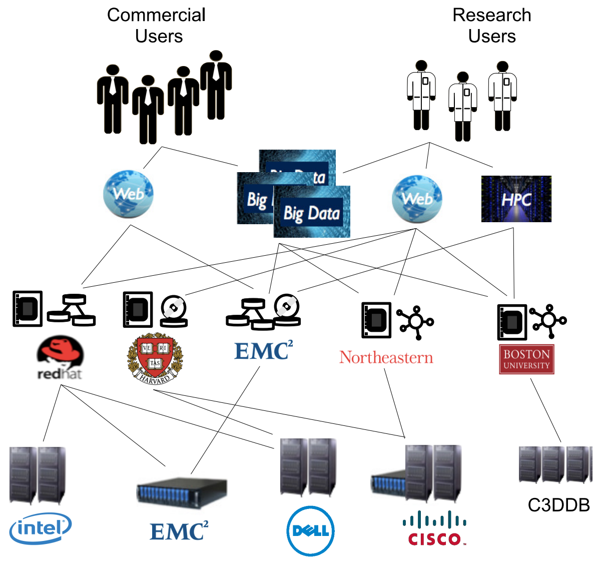

Fig. 2: The Massachusetts Open Cloud

- Enable industry to operate hardware and software services and expose them to MOC clients.
- Enable researchers and OpenStack developers to perform at scale experiments in cloud computing.
- Provide an environment to host large public data sets, including the data sets of the Commonwealth.

Figure 2 illustrates an example operating model of the MOC Open Cloud eXchange. In this example, the lowest layer comprises Intel, EMC, Dell, Cisco and C3DDB[2] acting as hardware providers. At the next layer, exchange service providers (XSPs) provide different IaaS services. In this example, Red Hat manages Nova (compute) and Cinder (storage) services, using underlying hardware from in this case Intel, Dell, and EMC. EMC provides branded storage services—Swift (object storage) and Cinder (block storage)—exploiting characteristics of their own hardware. Research IT organizations at Harvard, Northeastern and Boston Universities each provide IaaS services tailored to the needs of technical computing users at their respective institutions. Each institution will have its own security model, policies, administrative staff, while maintaining control of its own operations, and determine which resources it will make available to what parties.

Most users access the IaaS layer indirectly by using intermediaries. These might include HPC portals, Big Data platforms (*e.g.*Hadoop, and Spark), or various web environments (*e.g.*PaaS environments like CloudFoundry, or OpenShift), and may be deployed and managed by industry partners or by the universities. These services will typically be implemented on top of resources provided by IaaS service providers, choosing these resources based on factors such as performance, price, and usage policies. For example, a life sciences Galaxy[12] service for genetic sequencing may make use of C3DDB services for most operations, but request services from commercial providers when load is high or when handling requests from users not authorized to make use of C3DDB resources.

[2]A grant-funded limited-access resource managed by some of the MOC institutions and restricted to life sciences users.

V. Supporting the Requirements

Features needed to support an OCX include mechanisms for (i) service discovery and (ii) requesting and configuring services; (iii) a cross-provider authentication and authorization mechanism which allows services to interact in a secure manner, (iv) consistent naming to allow interaction between services from different providers, and (v) an inter-provider communication mechanism with sufficient bandwidth. This section describes these features and points out how they differ or extend the architectural features provided by OpenStack.

A. A Global Service Directory

In OpenStack the types of services are limited, and there is a single endpoint to request services of a particular type. In the OCX model there is no bound on the number of service endpoints or service types; therefore a cloud-wide service directory is used for service location, in which providers advertise their services and features, and users in turn locate these services.

The directory provides attributes and location of services in a machine-readable form, which may then be used by client scripts or intermediaries which act on a client's behalf to select which services to use. Service attributes in this directory such as price, SLA, and security level must have unambiguous cross-provider definitions, while allowing for specification of parameters which may be variable (e.g. price, performance) or intangible (e.g. brand). The format is extensible, allowing the description of new services and attributes which may be introduced after the directory is deployed. For example, it must allow the specification of Hardware as a Service (HaaS, Section V-H), higher-level services such as Hadoop, or more modest variations such as changes in API versions.

This new Global Services Directory makes use of OpenStack's simple service directory (in Keystone) that maintains the list of endpoints for each of the OpenStack services in a single region. Each provider is assumed to employ OpenStack so information can be extracted from each of these per-provider directory, and used to populate the cloud-wide service directory together with extra meta data describing the characteristics of the service in the new service description language we are developing for the OCX service directory.

B. Client Controlled Selection of Services

In an OCX the client should be able to control which of many providers and versions of a service will be used to handle her requests. To provide this control, a customizable scheduling and resource allocation library is used. This library can be incorporated into intermediaries and end-applications to support modified or extended scheduling policies. This library does not replace the existing OpenStack mechanism but is instead layered above it, selecting which provider to use, after which the existing per-provider scheduler determines which specific service will be used to handle the request.

This approach is a fundamental departure from the OpenStack model of one API endpoint for each type of service, with all selection performed by a scheduler behind this endpoint considering only resource availability and a small number of user-provided constrains. Extending the OpenStack scheduling model to an OCX would be problematic, as not only does the scheduling problem become more complex as the set of features to be selected from grows, but this shared scheduler must be trusted to fairly and accurately distribute requests across providers.

C. Multi-Provider User Interface

The user interface for a client using the OCX has several key requirements. The first is the ability for a client to use one interface to access offerings from multiple providers. For example, a client can select virtual machine compute from Northeastern and storage from EMC (Figure 2). In addition, the interface must be extensible so that new untrusted interfaces can be exposed to users in a seamless fashion. For example, a customer interacting with a Spark big data service, stood up by one provider, uses a single management panel to access that and other services that she chooses to use. Finally, the UI backend should not be a shared service that can cause one tenant to impact the security or performance of other tenants.

The first requirement means that the GUI exposes a marketplace of services in similar fashion to any app store, where the customer can shop for different services. The GUI collects the information from the OCX services directory and exposes them directly to customers.

For the second requirement, extensibility is enabled by allowing services to provide their own GUI displayed in an inline frame; common styling is being developed to better integrate these “plug-in” UIs. The OCX GUI directly interacts with only a small set of OpenStack services (Nova, Neutron, Cinder) using the OCX library to provision core IaaS services.

For the third requirement, a new instance of the GUI is provisioned in client-controlled capacity, avoiding dependency on services from any one provider. This differs from OpenStack, where the Horizon GUI is a shared web-based service from the single provider.

D. Inter-provider Interactions

A primary goal of the Open Cloud Exchange model to allow tenants to “mix and match” service components from different providers. At the infrastructure level, this means being able to combine compute services from one provider with storage services from one or more separate providers. Doing so requires a significant change from today’s model, as efficient and secure provider-to-provider interactions are required. Rather than requiring extensive trust between providers, we instead require authorization of the request from the end user.

There are two issues that must be addressed. One is a naming issue to allow resources in one service provider to be able to name the service in another provider. Another issue is trust or lack of trust, to ensure that a service in one provider that is used on behalf of a service in a second provider has the appropriate restrictions.

Resource addressing: In OpenStack each resource object is identified by a UUID; the type of the object, and thus the endpoint which may operate on it, is implied by context. In contrast, resource objects in an OCX system will be identified

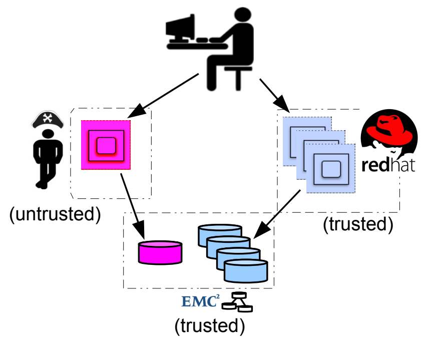

Fig. 3: Secure delegation: ensuring that untrustworthy compute provider A only accesses storage volumes (red) accessed by VMs on Provider A

by both UUID and the URI of the service endpoint [3], allowing a subsystem, *e.g.*Nova, to route requests to destinations on other providers.

Trust delegation: In a multi-provider cloud it is important to limit the scope of trust delegation when a client invokes a service that must interact with other services on behalf of that client. In the example shown in Figure 3, an untrusted compute provider (red) hosts a user’s virtual machine and accesses a storage volume on a third-party storage provider. The same user allocates other, more sensitive virtual machines (blue) on a more reliable compute provider, with separate storage volumes hosted by the same storage provider. Per-object access control is needed to ensure that the red provider can only access the red storage and not the blue storage. Work is underway with the Keystone team to add mechanisms[4] which will allow the required level of fine-grained access delegation.

E. Federated authentication and authorization

It is necessary in an OCX for each provider to be able to have their own authentication and authorization services to support their own adminstrators and users. On the other hand a global authentication and authorization service is needed to allow a client to concurrently use multiple services (that may in turn invoke other services) from different providers. Consequently per-provider services must federate with the OCX service.[5]. Explicit sharing agreements dictate data elements such as domain and project definitions that need to be common across two or more keystone instances owned by different providers.

For example, in Figure 2 Harvard might be unwilling to trust a service stood up by Northeastern for authorizing its administrators to access a Harvard-managed Nova service. At the same time, we want customers to the cloud to be able to be able to log in once and access services from all the providers.

F. Network spanning and security of underlying protocol

Any OCX solution requires establishing network connectivity between services deployed on different providers. In the

[3]We are currently working with the OpenStack community on a specification, see https://review.openstack.org/#/c/132623/
[4]see https://review.openstack.org/#/c/123726/
[5]see https://review.openstack.org/#/c/123782/

near term, simple tunneling via a gateway server (normally a VM) between each provider will be used. In the long run, however, better mechanisms will be needed, as data centers often have massive bisectional bandwidth and such gateway servers will introduce severe bottlenecks. Also, the simple tunnelling solution will not support rich networking functionality, such as is becoming common with network function virtualization.

In the long run, SDN (Software Defined Network) functionality will be developed to provide the connection between the service providers. The SDN's for each provider negotiate efficient distributed paths and provide rich network functions. This is an new area of research as to-date nearly all the development has been on north and south interfaces for networking controllers, i.e. up from the compute resource to the top-of-rack switch and either further up or back down.

G. Securing underlying protocols

In addition to control interfaces, there are also underlying protocols used for the data paths, and in an OCX these have to be more secure than the protecols used in today's (e.g., OpenStack) single provider cloud (e.g. NFS and iSCSI). In the short term, network isolation can be used to establish pair wise isolated relationships when existing non-secure protocols are used. For example, a Cinder service will have a seperate isolated network for each Nova it supports, and volumes being exposed to a particular Nova service will only be exposed over a network shared with that Nova service.

H. HaaS

In a typical OpenStack cloud, one set of administrators controls all the hardware in the entire cloud whereas in an OCX, the hardware is just a service upon which other services can be layered. Sharing of physical resources in OCX is via Hardware as a Service (HaaS) [14] (Figure 4). The main difference between HaaS and related systems such as Emulab [1], Maas [6], and others [2, 8, 20] is that, rather than providing (and requiring) a single solution for both scheduling and OS provisioning, HaaS enables the use of arbitrary pre-existing schedulers and provisioning systems as well as its own.

HaaS uses its knowledge of compute nodes and network connectivity to enforce isolated configurations. To accomplish connectivity, it maintains a list of network switch ports, the NICs of compute nodes, and how the two are connected. Switches are supported via model-specific drivers. Currently VLANs are supported for connecting and isolating networks, though the model could accommodate other mechanisms such as OpenFlow.

In the example shown in Figure 2, Harvard and Red Hat both operate Nova instances, each using Dell resources, and (within the limits of the underlying resources) able to expand and contract their usage as needed. Such flexibility in redeployment of resources is key to the efficient utilization of cloud hardware; indeed, VCL [9] demonstrated major improvements in utilization and economics by being able to shift resources between a Cloud and HPC clusters.

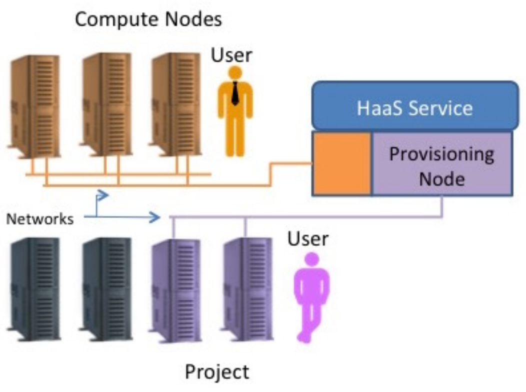

Fig. 4: Hardware as a Service (HaaS)

VI. Concluding remarks

Support for multiple, mutually distrustful, service providers to offer and deliver services to clients in the same cloud requires a set of architecture features that are a departure from the usual cloud support architecture. These necessary architecture features described, while critical for an OCX, have value even if a single provider is setting up the cloud. For example:

- The global service directory, in conjunction with per-user scheduling, reduces barriers to offering rich heterogeneous services, especially those targeted to smaller fractions of the user base.
- The HaaS layer can be used by other services that require bare-metal access, such as OS distributions requiring direct access to non-virtualized hardware such as GPUs. (e.g. Neurodebian[13])
- Federated authentication and authorization mechanisms may be used in hybrid private/public OpenStack clouds.
- Architectural features that eliminate (or minimize) trust between services will provide deeper security guarantees even in a single-provider cloud.

The OCX model will enable innovation by creating a level playing field where many different providers can cooperate and compete in a shared cloud. No single provider has competitive advantage since no provider owns the underlying infrastructure services. At any layer, providers that can innovate in just one area are free to participate, reducing the barrier to entry. At the higher level, new services will have richer infrastructure they can select between and are first class citizens in the cloud. Clients can easily move between different compatible services, reducing vendor lock in. Competition will encourage transparency, which will in turn lead in turn to higher efficiency.

Although there is still trust needed by the clients and the operator of the exchange, the shared services are few and simple (relative to a full cloud). From a security perspective, we believe that most of the risk remains with the providers standing up cloud services. A client can choose which underlying providers to trust, and can make her own tradeoffs of performance versus security.

The changes outlined are significant and will require broad community involvement. However, the changes are, for the most part, consistent with directions the community is already going.

Acknowledgments

Partial support for this work was provided by the MassTech Collaborative Research Matching Grant Program, National Science Foundation awards #1347525 and CNS-1149232, and several commercial partners of the Mass Open Cloud who may be found at http://www.massopencloud.org. We are also grateful to the many contributors towards the MOC project, including Azer Bestavros, Ian Denhardt, George Silvis III, Zhaoliang Liu, Andrew Mohn, Jonathan Bernard, Valerie Young, Jonathan Bell, Rohan Garg, Ritesh Singh, Abhishek Raju Charmarthi, Ryan Abouzahra, and Viggnesh Venugopal.

References Cited

[1] D.S. Anderson, M. Hibler, L. Stoller, T. Stack, and J. Lepreau. Automatic online validation of network configuration in the Emulab Network Testbed. In *ICAC*, pages 134–142, June 2006.

[2] Sam Averitt, Michael Bugaev, Aaron Peeler, et al. Virtual computing laboratory (VCL). In *Proceedings Int'l Conf on the Virtual Computing Initiative*, pages 1–6, 2007.

[3] Azer Bestavros and Orran Krieger. Towards an open cloud marketplace: Vision and first steps. Technical Report BUCS-TR-2013-019, CS Department, Boston University, October 2013.

[4] Secure cloud bridging. Technical Report GA-AG-489-01, Brocade Communications Systems, Inc., San Jose, CA, 2013. available from www.brocade.com.

[5] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[6] Canonical. Metal as a Service. http://maas.ubuntu.com/.

[7] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. How to enhance cloud architectures to enable cross-federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 337–345. IEEE, 2010.

[8] Jeffrey S Chase, David E Irwin, Laura E Grit, Justin D Moore, and Sara E Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proc. IEEE HPDC*, pages 90–100. IEEE.

[9] Patrick Dreher and Mladen Vouk. Integration of high-performance computing into a VCL cloud. In *Proceedings of the 8th Workshop on Virtualization in High-Performance Cloud Computing*, VHPC '13, pages 1:1–1:6, New York, NY, USA, 2013. ACM.

[10] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

[11] GENI. http://www.geni.net.

[12] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, August 2010.

[13] Yaroslav O Halchenko and Michael Hanke. Open is not enough. let's take the next step: an integrated, community-driven computing platform for neuroscience. *Frontiers in neuroinformatics*, 6, 2012.

[14] Jason Hennessey, Chris Hill, Ian Denhardt, Viggnesh Venugopal, George Silvis, Orran Krieger, and Peter Desnoyers. Hardware as a service - enabling dynamic, user-level bare metal provisioning of pools of data center resources. Waltham, MA, USA, September 2014.

[15] Qin Jia, Robbert Van Renesse, and Hakim Weatherspoon. Supercloud: economical cloud service on multiple vendors. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 58. ACM, 2013.

[16] Rob Lloyd. Introducing Cisco's Global Intercloud, March 2014. In blogs@Cisco, available from blogs.cisco.com.

[17] The OpenStack Foundation. OpenStack: open source software for building private and public clouds. http://www.openstack.org.

[18] RedHat. CloudForms. http://www.redhat.com/en/technologies/cloud-computing/cloudforms.

[19] Wei-Tek Tsai, Xin Sun, and J. Balasooriya. Service-oriented cloud computing architecture. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 684–689, April 2010.

[20] Devananda Van der Veen et al. Openstack ironic wiki.

[21] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *HotCloud*, HotCloud'10, page 1010, Berkeley, CA, USA, 2010.