# Rethinking Cloud Platforms: Network-aware Flexible Resource Allocation in IaaS Clouds

Juliano Araujo Wickboldt
Lisandro Zambenedetti Granville
Federal University of Rio
Grande do Sul, Brazil
{jwickboldt, granville}@inf.ufrgs.br

Fabian Schneider
NEC Laboratories Europe
Heidelberg, Germany
Fabian.Schneider@neclab.eu

Dominique Dudkowski
Robert Bosch GmbH
Schwieberdingen, Germany
dominique@dudkowski.eu

Marcus Brunner
Swisscom Ltd
Switzerland
marcus@brubers.org

***Abstract*—Most of the current platforms for cloud infrastructure management are designed to deal mainly with computing and storage resources. However, when deploying highly distributed applications with strict network requirements, such as low delay or bandwidth guarantees, the support for specification and configuration of such requirements still lacks. Moreover, resource allocation strategies and algorithms are usually hard-coded into the cloud platform's core, making it very difficult to improve or adapt these strategies to better fit individual application and environment needs. In this paper, we introduce a new approach to cloud platform design, emphasizing three main aspects: (*i*) robust networking for coupling cloud computing with modern network paradigms, (*ii*) specification of complex virtual infrastructures, including network topology and application requirements, and (*iii*) programmability via an API to ensure customization at the core of the platform's resource allocation and optimization strategies. We still present a proof of concept prototype that we have implemented and deployed over a modern network testbed and also evaluated on an emulated network using Linux virtualization containers, Open vSwitch, and mininet. Using our prototype, we have conducted a case study featuring an information-centric networking (ICN) application. Initial results show the feasibility of our approach and the application deployment statistics.**

## I. Introduction

Current software platforms that enable Infrastructure as a Service (IaaS) cloud environments (*e.g.*, OpenStack [1], Eucalyptus [2], OpenNebula [3]) tend to separate resource management into computing, storage, and network. Computing management is tightly related to handling virtual machines (VMs). In this case, cloud resources mainly encompass volatile memory and processing power. Storage management, on its turn, enables allocation of large amounts of persistent data volumes – possibly distributed over many physical machines – and association of these volumes with VMs.

As opposed to computing and storage management, which have been extensively exploited in the last years, network management in cloud environments is rather in its early stages. Managing a network in a cloud includes enabling proper means for virtual components communication. Typically, these components are virtual interfaces of VMs, which can be created, destroyed, or migrated at any time, in a very dynamic fashion. Because network management is still very basic in cloud environments, most cloud management platforms rely on external management systems for any network-layer configuration (*e.g.*, DHCP servers, manual VLAN establishment, NAT or forwarding rules on `iptables`). This scenario configures the first drawback we intend to tackle in this paper, namely the rather rudimentary support for network management in cloud management platforms.

Another important design aspect that is present in most of the current cloud platforms regards the black-box-like centralized controller, which resembles cluster task schedulers. Based on the very definition of cloud computing, customers should describe the resource they need in a rather simple way, and receive back such resources, somehow provisioned by the cloud controller. More specifically, two main shortcomings exist here: (*i*) customer requirements are expressed in a too simplistic way; and (*ii*) cloud operators have very few opportunities in influencing and controlling resource allocation. As a result, some modern applications – particularly highly distributed and network intensive ones – that could benefit from cloud environments, are inhibited to exploit them because of the poor support to specify their strict requirements. With regard to (*ii*), influencing resource allocation in a way that optimizes the overall use of physical resources is only possible if key performance indicators of the network resources can be specified and exploited in an agile resource allocation process. This lack of flexibility, in terms of both requirement expressiveness and operator ability to influence in resource allocation, is the second problem tackled in this paper.

Given the aforementioned shortcomings, our main contribution within this study is the new and more holistic approach to cloud platform design. To this end, we propose and implement a *hybrid* cloud platform solution in which by *hybrid* we emphasize that network resources are considered explicitly as part of the cloud platform along with the traditional computing and storage resources. Our approach specifically stresses the following items: (*i*) more robust networking support for cloud environment through the use of modern networking paradigms (*e.g.*, software-defined networks (SDN)); (*ii*) support for richer specification of complex virtual infrastructures, including all kinds of virtual resources (*i.e.*, computing, storage, and network) and application-specific requirements (*e.g.*, elasticity rules); and (*iii*) more flexible resource allocation strategies, with a programmable API, in such a way that operators can easily describe and run personalized algorithms for application deployment and optimization over the cloud.

To support our work, a prototype has been implemented to show the feasibility of our approach. This prototype was implemented and deployed over NEC's Advanced Network Testbed (ANT), which offers state-of-the-art technology in

978-3-901882-50-0 ©2013 IFIP

SDN with OpenFlow [4] and also evaluated on an emulated environment, using Linux virtualization containers with LXC [5], Open vSwitches [6] running also OpenFlow, and Mininet [7]. An information-centric networking (ICN) application, based on the NetInf [8] platform, has been deployed using our prototype, as a case study. Results show a promising path towards end-to-end deployment – from detailed specification and resource allocation to monitoring and adaptation – of network-intensive applications in cloud environments.

The remainder of this paper is organized as follows. In Section II, we describe some of the currently available cloud platforms and their main features. In Section III, the conceptual architecture that organizes modules and their interactions of our solution is introduced. The prototype developed as a proof of concept is detailed in Section IV. A case study featuring the deployment of an ICN application is presented in Section V. Finally, in Section VI, we conclude this paper with final remarks and future work.

## II. Related Work

In this section we examine the main features of recent cloud platform solutions and to what extent they provide support for network control and management. Moreover, we analyze, from the flexibility point-of-view, how well the specification of application requirements are supported and how the operator can influence the allocation of resource on such platforms.

Vaquero *et al.* [9], for the first time, provided a consolidated definition of the cloud computing concept firstly analyzing its essential properties based on a large number of previously given definitions. That study reveals that the traditional approach to cloud computing emphasizes much more the provisioning of virtual computing and storage resources, while network is considered a means of connecting those virtual resources together. Benson *et al.* also highlighted some of the limitations of cloud platforms in the support of robust networking functions.

Taking as examples some major open source platforms currently available, namely, Nimbus [10], OpenNebula [11], Eucalyptus [12], and OpenStack [1], it is possible to conclude that network configuration is still pretty basic in most of them. One aspect that is present in most network configuration modes of these platforms is the use of DHCP servers to configure IP addresses for virtual machines. The procedure is usually based on defining different IP ranges of sets of virtual machines (*i.e.*, belonging to different clients). Eucalyptus additionally provides a mode where it is possible to use VLAN tags to isolate traffic between sets of virtual machines, for security reasons. Also, assignment of public IP addresses for remote access to virtual machines and configuration of filters (*e.g.*, allow only HTTP traffic to a specific public IP) is possible in Eucalyptus.

From this current scenario, it is possible to observe that the network in these cloud platforms is only meant to provide flat IP connectivity between virtual machines that belong to the same set, neglecting strict communication requirements such as bandwidth and delay guarantees. Moreover, assuming that not all virtual machines in the same set communicate to one another in the same way, the fact that there is no concept of virtual topology or virtual links between virtual network devices makes it hard to optimally place them within the datacenter.

Only recently, OpenStack started a parallel project called Quantum [13] to further extend its networking capabilities. This project aims to develop the concept of Connectivity-as-a-Service in this platform, adding transparent VLAN creation, explicit definition of network interfaces of virtual machines, and plugins to expose API extensions for more complex functionality support, such as network QoS. Although promising, this project is still in its early stages and, at the time of this writing, we still cannot fully analyze or evaluate its functionality. Nevertheless, this project emphasizes, once again, that network configuration and management in current cloud platforms is a secondary goal only dealt with based on plugins and extensions.

Another aspect of current cloud platforms that draws our attention regards the flexibility issues for both specification of the application that one needs to deploy and the resource allocation and optimization tasks. The current status of most of the platforms mentioned before concerning the application specification is as follows: the user informs the basic resource needs of his/her application (*e.g.*, how many virtual machine instances, how much RAM memory, how much disk space, which operating system image should be deployed), and the cloud controller allocates and maps virtual resources to physical ones. There is no means for the end-user to specify, for example, constraints for communication between virtual machines or rules for dynamic allocation of more/less resources.

Nowadays, many of the cloud platforms already support some interoperability interfaces, such as the Open Cloud Computing Interface (OCCI) [14]. By using such interfaces, one is able to remotely send requests directly to the platform to allocate virtual resources. However, that only eliminates the need to access the user interface in order to place requests. If one wants to really change resource allocation strategies on the core of a cloud controller, it would be necessary to dig into the source code – which is usually open – in order to re-implement it.

Based on this state-of-the-art, we conclude that a more integrated approach is needed when designing a cloud platform raising the need for network resources to the same level as computing and storage have always been. Also, more flexibility needs to be supported to enable better application specification and optimization of resource usage. Therefore, in the remainder of this paper we will present the concepts, implementation, and evaluation of our integrated cloud platform solution.

## III. Conceptual Architecture

In this section, we present the conceptual building blocks of our solution. This work is conducted in the context of the FP7 Scalable & Adaptive Internet soLutions (SAIL) project [15], more precisely under the work package of Cloud Networking (CloNe) [16]. Therefore, many concepts and assumptions we use in our work are aligned with the project's definitions.

The architecture depicted in Figure 1 is described below to aid on the understanding of the solution as a whole. We have named our solution and prototype (presented in Section

IV) Hybrid Flash Slice (HyFS) Manager. Slice is an already well established term in both computer and network virtualization environments. When we talk about "hybrid" slices, the meaning is related to the inclusion of different kinds of resources, which compose the infrastructure necessary to deploy applications over the cloud (*i.e.*, computing, storage, and network). By the term "flash" we refer to the dynamics of the slice, since it can be created, destroyed, expanded, and shrunk based on demand fluctuations on a time scale comparable to today's computing-centric cloud platforms.

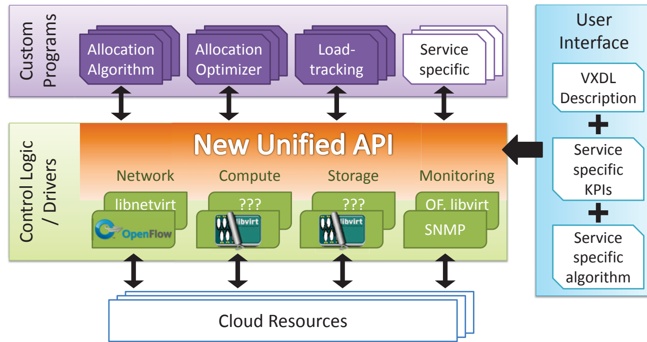


Fig. 1: Conceptual Architecture of HyFS Manager

Before diving into the architecture's components, it is necessary to clarify that we work with an Infrastructure as a Service (IaaS) scenario, where a user is interested in renting virtual resources to deploy his/her application. The cloud provider owns the resources and is interested in providing them to many users in the most cost effective way. Other complex business models for clouds have been discussed in previous studies [17]. Following, we describe one by one the major components of our architecture.

### A. *User Interface*

The interaction with the HyFS framework, for the human users, is performed through the *User Interface*. Using this interface, the user sends a request for a set of resources in the form of a virtual infrastructure specification of a HyFS to be deployed by the cloud provider. This specification contains a *description* – in general, an XML file – providing details on the resources needed for a given service or application. Optionally, the specification can also contain information about *service specific key performance indicators (KPIs)* and *service specific resource management algorithms*.

*Description* provides the system with more or less information about the initial allocation needs for an application, depending on the previous knowledge the user has about it. In other words, this is a trade-off where if the user informs very few characteristics about the virtual infrastructure, the system will have more freedom in deciding where virtual resources should be placed and how they should be connected to each other. On the other hand, when greater level of detail on the virtual infrastructure constraints and goals are available, the system only needs to look for an allocation that satisfies them.

*Service specific KPIs* are provided from within the application to the cloud and shall be monitored by the *Control Logic*. Considering the effects of the level of detail contained in the *description*, particularly when few details are provided for the initial allocation, it is probably a good idea to inform some useful KPIs to allow the post-optimization of a possibly poorly provisioned running application.

The HyFS framework contains a set of generic algorithms ready to be used for deployment and optimization of virtual infrastructures. Nevertheless, the specification of a HyFS can also inform *Service specific algorithms* to be used for deploying or adapting its own allocation changing conditions and utilization in more specific scenarios (see *Custom Programs* in the next section). These algorithms rely on the KPIs – either application specific or measured from the cloud infrastructure – in order to perform optimization actions. KPI thresholds could trigger actions, such as create/destroy/move virtual machines, expand link capacity, or create new connections.

Moreover, the *User Interface* provides basic functionality for the user and the cloud administrator to handle individual virtual resources, *e.g.*, creating virtual new storage volumes, migrating virtual machines, and establishing virtual links.

### B. *Custom Programs*

*Custom Programs* implement high level resource management functions in the form of algorithms. We envision that the HyFS framework will be shipped already with a basic set of programs. Moreover, these programs might be customized by the cloud provider to better fit the environment's needs (*e.g.*, centralized vs. distributed, complex multi-objective vs. simple heuristics). In order to implement these algorithms, operators will have access to a set of common functions provided by a *Unified API* implemented by the *Control Logic*. Also, as mentioned before, these functions can be service-specific. In this case, their development should be performed jointly with the user and then made available along with *Custom Programs* to be selected and used for a specific application.

The programs available in the framework are divided into four classes: (*i*) *Allocation Algorithms* are responsible for calculating a resource allocation based on the initial request; (*ii*) *Allocation Optimizers* continuously try to optimize the allocations of all active HyFSs based on overall goals, such as minimizing bandwidth or energy consumption, thereby correcting potentially non-optimal initial deployments; (*iii*) *Load-Tracking* will adapt the deployed virtual infrastructures based on events triggered by monitoring or input from the operator considering elasticity parameters of HyFSs; (*iv*) *Service specific* are allocation of optimization programs that are designed specifically to attend a demand of a particular application.

### C. *Control Logic / Drivers*

The main role of the *Control Logic* is to provide a technology independent interface for managing cloud resources. The *Unified API* enables access to information of *Network*, *Compute*, and *Storage*, as well as *Monitoring* of all three on the same level. Pluggable drivers for different technologies can also be added or removed depending on the needs of each cloud provider. *Network*, *Compute*, and *Storage* drivers are necessary to abstract the complexity of many possible different underlying virtualization technologies that might be found on modern data centers. These drivers implement particular low

level functions of each technology and expose higher level interfaces hiding, from the person who implements the algorithms, platform-specific parameters. There might be different drivers in the same category, for example, both OpenFlow or libnetvirt [18] can be employed for network control.

*Monitoring* abstracts the access to metrics from the application level (inside a HyFS), the virtual infrastructure level (*e.g.*, KPIs from virtual machines and virtual links), and also from physical resources (*e.g.*, average CPU usage of a host in the datacenter). Besides monitoring a variety of metrics from virtual and physical resources, it is also among the tasks assigned to the *Monitoring* module triggering events according to defined thresholds.

## IV. Prototype Implementation

As a proof the of concept, we have implemented a prototype following our new approach to cloud platform design. Most of the implementation work has been performed using the Python programming language. Additionally, several third party tools and APIs have been brought together to materialize the concepts presented in the architecture. The implementation aspects are described in this section, following the conceptual architecture presented in Figure 1.

For the *Initial Specification* of a virtual infrastructure, we have used an extension of the Virtual Infrastructure Description Language (VXDL) [19]. This language allows the detailed specification of virtual elements, such as machines (`vNode`), storage (`vStorage`), routers (`vRouter`), links (`vLink`), and access points (`vAccessPoint`). With these elements, one can create a complete virtual infrastructure, including the virtual network topology and its connections to the outside world. At this point in time, VXDL specification is being extended to support the representation of elasticity rules for the virtual infrastructure. These rules dictate which metrics should be considered in order to trigger actions to optimize the virtual infrastructure. So far, in the prototype, the user needs to prepare a VXDL file beforehand and then submit it to the system via its Web based *User Interface*. A designer component to help on creating this files could be valuable and are considered for future versions of the prototype.

The core of the cloud platform is coded in Python and implemented as a Web based application. We have employed a three-layered Model-Template-View (MTV) framework called Django [20]. The Template layer implements the presentation of the *User Interface* to the users of the system (*i.e.*, the end-user and the cloud operator). Through this interface users are able to perform many useful operations over virtual resources, such as checking virtual machine status, establishing virtual links, or requesting a whole HyFSs at once using a VXDL file.

In the View layer, the logic of the platform is implemented. Part of this logic includes basic functions for managing virtual and physical resources, such as starting/stopping/migrating virtual machines and configuring access to hypervisors on physical nodes. It is also at the View layer that we have implemented the base framework for the Custom Programs (*i.e.*, allocation and optimization algorithms) to run. When a HyFS needs to be deployed or optimized, one of the algorithms available for each purpose is selected to actually perform the resource allocation actions. At this stage of the prototype development, this selection is performed by the user who is deploying the HyFS. We plan to include, in the future, some intelligence in this component to dynamically choose a suitable algorithm for the deployment of a specific type of application.

The algorithms are implemented as Python code directly into the platform's core. For the person who designs these algorithms, the platform provides a very high-level object oriented API to perform resource management operations directly into the cloud. Some of the main operations available through this API are briefly described as follows:

**– Create/Destroy Virtual Machine:** defines a virtual machine (guest) on a node (host) of the cloud, including the transfer/deletion of the image file. If no particular node is specified, the platform will choose one according to its internal policies (*e.g.*, based on resource availability).

**– Start/Stop/Suspend/Resume Virtual Machine:** basic operations to handle the state of the guest.

**– Migrate Virtual Machine:** undefines a guest in one host and defines it on another. In this case, the destination host should be specified.

**– Create/Destroy Virtual Storage:** allocates/deletes image files on hosts.

**– Attach Virtual Storage to Virtual Machine:** attaches image files to a given virtual machine.

**– Create/Remove Virtual Link:** virtual links are created to connect point-to-point virtual network interfaces of virtual machines. For a future version, we consider creating virtual switches and connecting virtual interfaces to virtual switch ports.

**– Create/Remove Virtual Link Bundle:** creates/removes several virtual links in one operation. This is mostly useful to reduce communication overhead between the platform and the external network controller that actually establishes the links (*e.g.*, OpenFlow Controller).

**– Establish/Disable Virtual Link:** establishes/disables layer 2 links in the network. We chose to restrain to layer 2 connectivity to have minimum interference on the choices for guest operating systems and communication protocols. In other words, we do not want to impose that deployed applications run TCP/IP.

**– Discover/Monitor Physical Resources:** this is a collection of operations to discover hosts and network topology available on the datacenter. This collection also includes retrieval of several kinds of information about resource usage and allocation (*e.g.*, memory, processor, or disk usage, and network capacity/utilization).

One last fact about the algorithms is that, the consistency of the HyFS allocation as a whole is the responsibility of the algorithm. In other words, if the deployment fails in a certain point, the algorithm should catch the exception and implement remediation actions. Otherwise, the slice will be partially deployed.

The database that stores HyFS information is implemented as Django models. These models create an object-oriented

view of the deployed virtual infrastructures. The database is populated with information that comes from both VXDL files as input from the users, and can be obtained by querying the hypervisors. The models are persisted to a local SQL relational database by the Django framework.

The three components that implement different management concerns in our prototype are *Network Allocator*, *Compute Allocator*, and *Storage Allocator*. These components are divided each into two main parts: (*i*) an interface where they expose the management operations for the API, as mentioned before, and (*ii*) one or a set of adapters to implement the low-level functions in the underlying technology. In the network part, for example, we implemented an adapter to communicate with a POX OpenFlow controller [21] (Python implementation of NOX [22], which is much more efficient than using NOX with Python bindings) and to establish virtual links as OpenFlow rules. For storage and computing management, our current development is based on Libvirt (Python binding) [23], which implements communication with several different hypervisors.

Finally, our prototype has been developed and deployed on the Advanced Network Testbed (ANT), which is running within NEC Laboratories premises. The testbed essentially contains an array of standard physical machines (nodes). All machines are interconnected by two different local networks: the one for management traffic is a standard ethernet network, which is used by our prototype to communicate with the physical nodes in order to do the management operations (*e.g.*, start virtual machines or communicate with the OpenFlow controller); and the second, OpenFlow network runs experimental traffic where communication between virtual machines flows through the virtual links created by our prototype. We have tested our prototype with NEC's OpenFlow enabled IP8800 network switches that implement OpenFlow protocol version 1.0. Although we are able to run the prototype on the testbed directly on hardware, in order to obtain the results presented in this paper, we have reproduced an emulated infrastructure to be able to create slightly more complex topologies. Details on the setup of the experiments and the case study are presented in Section V.

## V. Case Study and Experiments

For evaluation purposes, we have used a case study based on the deployment of a NetInf [8] application using our cloud platform prototype. NetInf is a novel information-centric networking platform being developed also under the SAIL project. It encompasses routing information through a network based on the content itself, instead of relying on regular network addressing schemes. One of NetInf's assumptions is that in the future network devices will be shipped with native support to NetInf protocol. This assumption is obviously not yet satisfied by current networks, so NetInf applications will rely on cloud platforms that can meet their unusual communication needs for large scale deployment.

The scenario we consider for using the NetInf application is the following. A worldwide video producer (*e.g.*, TV channel) produces all kinds of video (*e.g.*, news, TV series, cartoons) in a central office. The producer company wants to stream its videos on demand over the Internet to people that may come from all countries at any time. NetInf routers are able to receive the request from the clients and route their requests through the network until it reaches a router that has the requested object. Once the object is found in the network, it is routed back to the client and cached by all routers in the path. With this caching system the next requests placed by clients for the same object may hit a cache along the path, avoiding the transmission again from the source and improving network utilization. The topology for this kind of content distribution network (interconnections of caches) should be a multi-level tree, as shown in Figure 2.

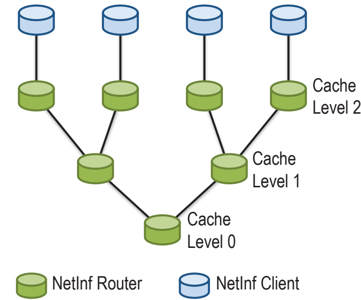


Fig. 2: NetInf application deployed topology

In this case study, we deploy virtual machines running NetInf routers as well as NetInf clients. In a real world application, clients would be expected to come from outside the cloud, but to simplify the scenario we just deploy them within the same slice as the routers. All 7 NetInf routers and 4 NetInf clients are deployed from a Debian Squeeze image file sizing near 330MB, expansible up to 500MB. Once deployed, the NetInf router in the root of the tree already has published all the objects available for download. The clients will start placing requests for random objects nearly every minute as soon as they start up.

For the deployment of the NetInf application, we have designed a simple algorithm that is efficient enough for the specific application at hand. This algorithm is divided into three main phases: (*i*) resource discovery, (*ii*) reasoning, and (*iii*) resource allocation. In the first phase, the algorithm makes use of the API to gather information about the physical resources available (*e.g.*, memory usage of hosts and network bandwidth allocations). After that, in the second phase, the algorithm computes where to place virtual machines based on very specific heuristics. Instead of trying to allocate one virtual machine at a time, we decided to allocate pairs of virtual machines based on the links that connect them. For example, let's consider a simple HyFS with 3 virtual machines, VM1, VM2, and VM3, connected by two links VM1↔VM2 and VM2↔VM3. This virtual infrastructure would be deployed in two pairs (VM1, VM2) and (VM2, VM3). When the first pair is being processed, both virtual machines are considered free to be placed anywhere in the cloud, so the algorithm will choose a host with low resource utilization (previous memory, CPU, and storage allocations) to deploy VM1 of the pair. Since there is a link between VM1 and VM2, we assume that these virtual machines will communicate often, therefore VM1 and VM2 should be placed close together in the network. Thus, for the placement of VM2, the algorithm will prioritize physical

hosts that have low resource utilization and are not too far away from the first one chose for VM1. We have created a metric that is calculated in each iteration for every host that is a good candidate to receive each virtual machine considering its resource utilization and distance on the network, but we will not describe the details of this metric because of space constraints and scope of the paper. For the next pair in our example, VM2 would be already placed, so the algorithm would only have to find a good host for VM3 and the HyFS would be ready for deployment.

In the last phase, the algorithm deploys all virtual machines on the hosts assigned in the previous phase. This deployment encompasses copying the virtual disk image, defining the virtual machine in the hypervisor, and starting it; these are operations all carried out by the API. Finally, with all the virtual machines in place, the algorithm establishes all the necessary virtual links in one single operation (virtual link bundle). We note that, although the algorithm considers resource availability – in terms of memory, CPU, disk, and network bandwidth allocation –, actual bandwidth control is still missing in the current version of our implementation, so the establishment of links is always successful. It is also important to emphasize that we do not claim that this is an optimal algorithm for resource allocation. Nevertheless, it has shown to be efficient enough for the deployment of our specific application and, more importantly, it was easily implemented in a few hours with our proposed API.

### A. Deployment Statistics

Before discussing the deployment statistics we need to introduce the emulated scenario prepared for the experiments. Figure 3 shows the datacenter topology we have created with 4 Open vSwtiches running OpenFlow protocol and being coordinated by the POX controller. Also, every one of the 5 hosts of the datacenter is emulated by a Linux virtualization container based on LXC. All this setup is started as a Mininet script on one physical machine where experiments were conducted, containing an AMD FX-8120 Eight-Core processor and 16GB of RAM memory. Our platform runs also on this physical machine and all LXC hosts run one instance of libvirt to receive commands from the platform.

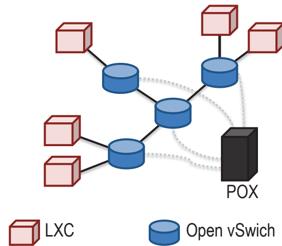


Fig. 3: Emulated physical topology based on LXC and Open vSwitches

We have monitored the allocation algorithm in each phase and gathered statistics from the deployment of the NetInf application, which encompasses creating 11 virtual machines and 10 virtual links (topology of Figure 2). The results achieved with the deployment of this scenario are presented in Table I. The deployment was performed 30 times, therefore the time in seconds presented is actually the average time for each specific operation performed. The number of experiments was determined considering the overall average time to execute all operations until the confidence interval was greater than or equals to 95%.

TABLE I: Deployment Statistics: Overall Time

| Info. gathering | Reasoning | Image copy | VM define | VM Start | Link estab. |
|---|---|---|---|---|---|
| 0.20s | 4.65s | 5.71s | 17.88s | 23.81s | 5.56s |

In Table I, it is remarkable that most of the time is actually spent on defining and starting virtual machines. Since our algorithm performs all operations sequentially, *i.e.* starts one virtual machine at a time, there exists an opportunity for improvement by making this process in parallel. It is important to notice that image copy operations are actually performed locally in this experiment (no data is transmitted over a network), so one cloud expect longer times in a production environment. Finally, link establishment time, which encompasses the communication between the platform and the POX OpenFlow controller via JSON Web services, is performed as a bundle link creation operation. Therefore, the controller itself will be responsible for finding the actual routes between virtual machines once traffic starts flowing.

## VI. Conclusion

In this paper we have discussed some of the current shortcomings of current available cloud management platforms. We have then proposed a new approach to cloud platform design emphasizing mainly: (*i*) robust support for network configuration and management based on modern networking paradigms, (*ii*) support for richer specification of complex virtual infrastructures, and (*iii*) programmability at the core of the platform via a high-level API.

The results obtained show a promising path towards the automated deployment of complex and mainly network intensive applications on top of a cloud infrastructure. We have presented the deployment of a complete ICN application based on the NetInf platform, including 7 routers and 4 clients in a little under 60 seconds of total deployment time. Besides instantiating a set of virtual machines, like most cloud platforms would support today, from a VXDL specification file, we also created a virtual topology for the application based on the configuration of virtual links using OpenFlow SDN technology. Our programmable API has shown to be easy to use allowing us to quickly develop an algorithm that works well enough for the deployment of the specific type of application we intended to use in our case study.

In future investigations we plan implementing more complex allocation algorithms, possibly running some pieces of these algorithms distributed on the cloud nodes to improve performance. We also plan to implement optimization algorithms to take advantage of the elasticity that the cloud environment can provide to applications. Finally, we intend to consider the multi-cloud provider scenario and coordinate the deployment of the HyFSs across several datacenters.

## REFERENCES


[1] Rackspace Cloud Computing, "Openstack cloud software," 2010, Accessed: Feb. 2012. [Online]. Available: http://openstack.org/

[2] Eucalyptus, "The open source cloud platform," 2009, Accessed: Feb. 2012. [Online]. Available: http://open.eucalyptus.com/

[3] OpenNebula, "The open source solution for data center virtualization," 2008, Accessed: Feb. 2012. [Online]. Available: http://www.opennebula.org/

[4] N. McKeown, T. Anderson, H. Balakrishnan *et al.*, "Openflow: enabling innovation in campus networks," *SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, March 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[5] LXC, "Linux Containers," September 2012, Accessed: Sept. 2012. [Online]. Available: http://lxc.sourceforge.net/

[6] Open vSwitch, "An Open Virtual Switch," September 2012, Accessed: Sept. 2012. [Online]. Available: http://openvswitch.org/

[7] Mininet, "Mininet: rapid prototyping for software defined networks," September 2012, Accessed: Sept. 2012. [Online]. Available: http://openflow.org/mininet

[8] D. Kutscher *et al.*, "D-B:1 The Network of Information: Architecture and Applications," EU FP7 Project Deliverable, Tech. Rep., July 2011, Accessed: Feb. 2012. [Online]. Available: http://www.sail-project.eu/wp-content/uploads/2011/08/SAIL_DB1_v1_0_final-Public.pdf

[9] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496100

[10] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," in *Cloud Computing and Its Applications (CCA)*, Chicago, USA, October 2008.

[11] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, Sept.-Oct. 2009.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, May 2009, pp. 124–131.

[13] Rackspace Cloud Computing, "OpenStack Quantum Project," 2011, Accessed: Apr. 2012. [Online]. Available: http://wiki.openstack.org/Quantum

[14] Open Grid Forum, "Open Cloud Computing Interface," 2012, Accessed: Sept. 2012. [Online]. Available: http://occi-wg.org/

[15] SAIL, "Scalable and Adaptive Internet Solutions," 2010, Accessed: Feb. 2012. [Online]. Available: http://www.sail-project.eu/

[16] P. Murray *et al.*, "(D-D.1) Cloud Network Architecture Description," EU FP7 Project Deliverable, Tech. Rep., July 2011, Accessed: Feb. 2012. [Online]. Available: http://www.sail-project.eu/wp-content/uploads/2011/09/SAIL_DD1_final_public.pdf

[17] A. Strømmen-Bakhtiar and A. R. Razavi, "Cloud computing business models," in *Cloud Computing for Enterprise Architectures*. Springer London, 2011, pp. 43–60. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-2236-4_3

[18] D. Turull, "libNetVirt: the network virtualization library," September 2012, Accessed: Sept. 2012. [Online]. Available: https://github.com/danieltt/libnetvirt

[19] G. Koslovski, S. Soudan, and P. Vicat-Blanc, "Modeling cloud computing and cloud networking with vxdl," in *World Telecommunications Congress 2012, Cloud Computing in the Telecom Environment workshop*, Miyazaki, Japan, Mar 2012.

[20] Django Software Foundation, "Django 1.4," Mar. 2012, Accessed: Apr. 2012. [Online]. Available: https://www.djangoproject.com/weblog/2012/mar/23/14/

[21] POX, "Pox openflow controller," 2012, Accessed: Sept. 2012. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[22] N. Gude, T. Koponen, J. Pettit *et al.*, "Nox: towards an operating system for networks," *SIGCOMM Computer Communication Review*, vol. 38, pp. 105–110, July 2008. [Online]. Available: http://doi.acm.org/10.1145/1384609.1384625

[23] Libvirt, "Libvirt: The Virtualization API - Version 0.9.9," January 2012, Accessed: Feb. 2012. [Online]. Available: http://www.libvirt.org