DOI 10.12694/scpe.v16i2.1089
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 2, pp. 187–204. http://www.scpe.org © 2015 SCPE

OPEN SOURCE SOLUTIONS FOR BUILDING IAAS CLOUDS

AMINE BARKAT[*], ALYSSON DINIZ DOS SANTOS[†], AND SONIA IKKEN[‡]

Abstract. Cloud Computing is not only a pool of resources and services offered through the internet, but also a technology solution that allows optimization of resources use, costs minimization and energy consumption reduction. Enterprises moving towards cloud technologies have to choose between public cloud services, such as: Amazon Web Services, Microsoft Cloud and Google Cloud services, or private self built clouds. While the firsts are offered with affordable fees, the others provide more privacy and control. In this context, many open source softwares approach the buiding of private, public or hybrid clouds depending on the users need and on the available capabilities. To choose among the different open source solutions, an analysis is necessary in order to select the most suitable according with the enterprise's goals and requirements. In this paper, we present a depth study and comparison of five open source frameworks that are gaining more attention recently and growing fast: CloudStack, OpenStack, Eucalyptus, OpenNebula and Nimbus. We present their architectures and discuss different properties, features, useful information and our own insights on these frameworks.

Key words: Cloud Computing, IaaS, OpenStack, CloudStack, Eucalyptus, OpenNebula, Nimbus

AMS subject classifications. 68M14

1. Introduction.

Open source cloud platforms were born as a response to the necessity of Infrastructure as a Service (IaaS) solutions to provide privacy and control over virtualized environments. Therefore the open source cloud platforms were primely used to build private clouds. Eventually, these open source solutions can be used to set up public clouds, private clouds or a mix of them, *i.e.*, hybrid clouds. With the emergence of different open source cloud solutions, the decision to choose the most suitable one becomes a confusing task, given the specific characteristics of each platform [18]. Moreover, since hybrid clouds are the most widely used nowadays, surveying open source middlewares that simplify cluster management is an important matter. In this context, several papers analyzed and compared different platform, trying to establish a starting point to look when deciding which open source cloud technology should be adopted.

Diverse researches detail comparison among cloud solution platforms. Table 1.1 shows related studies that compare cloud solutions, highlighting the analyzed solutions and their main focus/limitation.

Some studies are generalists on the approach and intend to provide a wide view on cloud solutions. Therefore they briefly present a higher number of solutions, with no comparison - the case in [33, 34] - or with a simple general feature comparison, as in [35].

More specific studies tend to deeper detail the analyzed solutions, considering a specific point of view. [11] focus on appearance design and novel features, while [32] focus on the scalability comparison of the platforms and [37] on placement policies, platform architecture and networking of the analyzed solutions. [13] details OpenStack and CloudStack solutions, but with deeper details only for OpenStack.

[28] provide a comparison of Eucalyptus, OpenNebula and Nimbus systems. The authors describe the different features and design, accurately highlighting tendencies in the focus of each of these products. Nevertheless, the study is outdated and missing two important open source solutions that gained importance since the publication of the paper, OpenStack and CloudStack.

In this context, this work is focused in IaaS open source cloud solutions. It presents in deep details the newest versions of OpenStack, CloudStack, OpenNebula, Eucalyptus and Nimbus and compare their general features and important properties, trying to provide useful information for users that need to choose an open source cloud software. This work is a continuation of the work done before [12], which compared only CloudStack and Openstack.

The paper is organized as follows: Sections 2, 3, 4, 5 and 6 describe respectively the architecture of OpenStack, CloudStack, Eucalyptus, OpenNebula and Nimbus and their important properties; Section 7 performs

[*]University of Bejaia in Algeria, Politecnico Di Milano in Italy (amine.barkat@polimi.it)
[†]Universidade Federal do Ceara in Brazil, Politecnico Di Milano in Italy (alysson@virtual.ufc.br)
[‡]University of Bejaia in Algeria, Telecom Sud Paris in France (sonia.ikken@telecom-sudparis.eu)

Table 1.1
Cloud solutions comparison studies

Study	Analyzed platforms	Main restriction
Voras et. al. (2011) [33]	OpenNebula, Eucalyptus, Ubuntu Enterprise Cloud, openQRM, Abiquo, Red Hat Cloud Foundations, OpenStack, Nimbus and mOSAIC	Brief introduction on any solution, no comparison provided
Zeng et. al. (2012) [34]	Amazon EC2, IBM smart cloud, Google App Engine, Windows Azure, Hadoop, Eucalyptus, OpenNebula and Nimbus	Brief introduction on any solution, no comparison provided
Endo et. al. (2010) [35]	XCP, Nimbus, OpenNebula, Eucalyptus, TPlatform, ECP (Enomalys Elastic Computing Platform) and Apaches VCL	Brief introduction on any solution, brief comparison provided
Amrani et. al. (2012) [11]	Eucalyptus, OpenNebula and Nimbus	Restricted to appearance design and new features description
Von Laszewski et. al. (2012) [32]	Eucalyptus, OpenNebula, Nimbus and OpenStack	Focus on scalability
Cordeiro et. al. (2010) [37]	XCP, Eucalyptus and OpenNebula	Focus on placement policies, platform architecture and networking
Baset (2012) [13]	OpenStack and CloudStack	Further details only on OpenStack
Sempolinsky and Thain (2010) [28]	Eucalyptus, OpenNebula and Nimbus	Outdated, missing OpenStack and CloudStack

comparisons between the platforms in terms of generalities, functionalities and proprieties. Conclusions are detailed on Section 8.

2. OpenStack. OpenStack is a cloud software that offers capability to control large pools of compute, storage and networking resources. It also empowers users providing on-demand resources [29]. Starting from 2010, OpenStack was developed by Rackspace Hosting and NASA [6] aimed to provide open source cloud solution to build public or private clouds. The mission of OpenStack is to enable any organization to create and offer cloud computing services running on standard hardwares. Provisioned as open source solution, OpenStack is built keeping these core principles in mind: *(i)* open source: all code will be released under the Apache 2.0 license allowing the community to use it freely; *(ii)* open design: every 6 months the development community hold a design summit to gather requirements and write specifications for the upcoming releases; *(iii)* open development: maintains a publicly available source code repository through the entire development process; and *(iv)* open community: produces a engaged development and user community through an open and transparent process.

The first version of OpenStack was entitled Austin and was released in October of 2010. Since then OpenStack adopts a policy of two major releases per year, totalizing 11 releases so far (the next one, entitled Liberty, expected to October of 2015). The Austin release of OpenStack had only the object storage (Swift) and compute (Nova) components, and presented important restrictions, like the support of objects limited to 5GB. The support to large files was introduced in the second release, entitled Bexar, together with the image registry and delivery service (Glance). New modules were introduced only on the fifth release, entitled Essex, with the graphical user interface (Horizon) and the security component (Keystone). The sixth release, Folsom, introduced the network as a core OpenStack project (then code-named Quantum, actually Neutron) and the block storage component (Cinder). Folsom was then, the first release to embody the OpenStack three more important modules, Swift, Nova and Neutron. Since this version, all the existent modules were fixed and improved and new shared services were added, such as: metering (Ceilometer) and orchestration (Heat) on the eighth release, Havana; the database service (Trove) on the ninth release, Icehouse; the data processing (Sahara) on the tenth release, Juno; and the bare metal service (Ironic) on the newest version, Kilo. The following Section 2.1 details the components that compound the general architecture of OpenStack.

2.1. General Architecture. As in any cloud platform, the infrastructure underneath OpenStack is standard hardware, which can contain any pieces of physical devices such as servers, disks or network devices. In order to provide cloud services, OpenStack develops virtualization layers, promoting the abstract view of physical infrastructure to end users. These virtualization layers are built up in a multicomponent architecture.

The OpenStack architecture consists of three main components: Compute (Nova), Network (Neutron) and Storage (Swift). Beside these three pillars, OpenStack has been developing many other services, each of those designed to work together to provide a complete IaaS solution. The integration of these services is facilitated through public application programming interfaces (APIs) offered by each service [6].

In the following, the detailed description of each component is provided.

2.1.1. Compute (Nova). The Compute component, codenamed Nova and written in Python, is the computing fabric controller responsible for managing large networks of virtual machines (VMs), and eventually to properly schedule VMs among available physical machines (PMs) [6]. Compute is a distributed application that consists of six components: Nova-api, Message Queue, Nova-Compute, Nova-Network, Nova-Volume and Nova-Scheduler. Nova supports the complete life-cycles of an instance in the cloud, starting from the request to initialize a VM until its termination. It follows this architecture:

- Nova-api: accepts and responds to end user compute API calls. Beside providing its own OpenStack Compute API, Nova-api is compatible with Amazon EC2 API, offering the potential to integrate with Amazon cloud services. It has another special Admin API reserved for privileged users to perform administrative actions. This component also handles the orchestration activities, such as: running an instance and the enforcing policies (*e.g.* quota checks).
- Nova-compute: is primarily a worker daemon that creates and terminates VM instances via hypervisor APIs. In order to do so, it accepts actions from the queue and performs system commands to fulfill them, while updating the database state accordingly. OpenStack supports several standard hypervisors (listed in Section 7) while keeping the openness that allows to interface other hypervisors through its standard library.
- Nova-volume: manages the creation, attachment and detachment of persistent volumes to compute instances. There are two types of block devices supported by a VM instance: (1) Ephemeral storage, which is associated to a single unique instance. One ephemerally stored block device life-cycle is coupled with the instance life-cycle, *i.e.*, when the instance is terminated, data on this storage will also be deleted; (2) Volume storage is persistent and independent from any particular instance. This storage can be used as external disk device where the data stored on it still remains even when the instance is terminated.
- Nova-network: is a worker daemon that handles network-related tasks. It accepts and performs networking tasks from the queue. Task examples include setting up bridging interfaces or changing iptable rules.
- Nova-schedule: handles the scheduling of VMs among PMs. While the scheduling algorithms can be defined by users, Nova-schedule supports by default three algorithms: (1) Simple: attempts to find least loaded host, (2) Chance: chooses random available host from service table, (3) Zone: picks random host from within an available zone. By allowing users to define their own scheduling algorithms, this component is important for building fault tolerant and load-balanced systems.
- Queue: provides a central hub for passing messages between daemons. This is usually implemented with RabbitMQ, but can support any AMPQ message queue.
- Database: stores most of the build-time and run-time state of a cloud infrastructure. For example, it provides information of the instances that are available for use or in use, networks availability or storage information. Theoretically, OpenStack Nova can support any SQL-based database but the most widely used databases currently are sqlite3, MySQL and PostgreSQL.

All the components of the OpenStack architecture follow a shared-nothing and messaging-based policy. Shared-nothing means that each component or each group of components can be installed on any server, in a distributed manner; while the messaging-based policy ensures that the communication among all components is performed via Queue Server.

2.1.2. Network (Neutron). The Network component of any cloud platform has important attributions: *(i)* to offer accessibility to resources and services; *(ii)* to provide address binding between different services (essential to support multi-tier applications) and *(iii)* to automatically configure the network (fundamental in auto-scaling scenarios).

The OpenStack Networking component gives operators the ability to leverage different network technologies to power their cloud networking. It does so through a rich set of APIs, multiple networking models (*e.g.*, flat or private network) and flexible plug-in architecture. Especially, the plug-in architecture - with the plug-in agent - enables, not only the usage of various network technologies, but also the ability to handle user workloads. In OpenStack, at network level, developers can implement their own load balancing algorithms and plug it in the platform to achieve better workload control.

The Network architecture consists of four distinct physical data center networks:

- Management network: used for internal communication between OpenStack components. The IP addresses on this network should be reachable only within the data center.
- Data network: used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the OpenStack Networking plug-in in use.
- External network: used to provide VMs with Internet access in the deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet.
- API network: exposes all OpenStack APIs, including the OpenStack Networking API, to tenants. The IP addresses on this network should be reachable by anyone on the Internet.

2.1.3. Storage. The Storage component, one of three main pillars of OpenStack architecture, is used to manage stored resources. OpenStack has support for both Object Storage and Block Storage, with many deployment options for each, depending on the use case.

Object Storage (codename Swift) is a scalable object storage system. It provides a fully distributed, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving, and data retention [6]. In Object Storage, data are written to multiple hardware devices, with the OpenStack software responsible for ensuring data replication and integrity across clusters. Object storage clusters are scaled horizontally while adding new nodes. If a node fails, OpenStack replicates its content from other active nodes. Because OpenStack uses software logic to ensure data replication and distribution, inexpensive commodity hard drives and servers can be used instead of expensive equipments. Therefore, Object Storage is ideal for cost effective, scale-out storage [6].

Block Storage (codename Cinder) allows block devices to be exposed and connected to compute instances for expanded storage, better performance and integration with enterprise storage platforms, such as NetApp, Nexenta and SolidFire [6]. By managing the storage resources in blocks, Block Storage is appropriate for performance sensitive scenarios, such as: database storage, expandable file systems, or the provision of a server with access to raw block level storage.

2.1.4. User interface - Dashboard. The OpenStack dashboard is a graphical user interface, to both administrators and users, that controls compute, storage and networking resources. Through the dashboard, administrators can also manage users and set limits on resources access for each user.

2.1.5. Shared Services. OpenStack Shared services are a set of several services that span across three pillars of compute, storage and networking, facilitating the cloud management operations. These services include the identity, image, telemetry, orchestration and database services [6]:

Identity Service (code-named Keystone) is the security service to protect resources access and usage. This service provides a central directory management, mapping users to OpenStack accessible services. It acts as a common authentication system across the cloud operating system. It also supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style logins.

Image Service (code-named Glance) is the repository for virtual disk and server images used by the VMs. In OpenStack, user can copy or snapshot a server image and immediately store it away. Stored images can be used as a template to get new servers up and running quickly and consistently.

Telemetry Service (code-named Ceilometer) aggregates resources usage and performance data of the services deployed in OpenStack cloud. This capability provides visibility into the usage of the cloud infrastructure and allows cloud operators to view metrics globally or individually.

Orchestration Service (code-named Heat) is a template-driven engine that allows application developers to describe and automate the deployment of the cloud infrastructure. It also enables detailed post-deployment activities of infrastructure, services and applications.

Database Service (code-named Trove) allows users to utilize the features of a relational database. Cloud users and database administrators can create and manage multiple database instances as needed.

2.2. Properties. Provisioned as IaaS, OpenStack is built following an open philosophy: to avoid technology lock-ins by not requiring specific technologies and to provide user freedom to choose the best slot that matches its needs [6]. In this section, we will analyze some important properties of OpenStack.

- Live migration: OpenStack supports two types of live migration: shared storage based and block live migration. The former supports live migration scenarios where the source and destination hypervisors have access to the shared storage, while the latter does not require shared storage.
- Load balancing: OpenStack supports load balancing at different scales. First of all, the supporting feature of live migration has enabled system administrators to distribute application workloads among physical servers by means of adjusting VM placement. Moreover, it is possible to control application workloads at VM level, service provided by OpenStack Network layer, controlled by Neutron component. This component, with a flexible plug-in architecture allows the development of run-time custom algorithms to distribute workloads among VMs. Indeed, OpenStack has an on-going project called Load Balancing as a Service (LBaaS) that is aimed to provide load balancing service to end users. This service has monitoring feature to determine whether the VMs are available to handle user requests and take routing decisions accordingly. Several routing policies are supported, such as: round robin (*i.e.*, rotates requests evenly between multiple instances), source IP (*i.e.*, requests from a unique source IP address are consistently directed to the same instance) and least connections (*i.e.*, allocate requests to the instance with the least number of active connections).
- Fault tolerance: Fault tolerance can also be handled at different levels, depending on the way the IaaS system is configured and deployed. At the VMs level, in order to prevent failures, users can develop scheduling algorithms (besides the three already supported algorithms by OpenStack) for placing the VMs that best fits to his use cases. Some scheduling algorithms have been designed at the present time, such as: group scheduling (*i.e.*, VMs that provide the same functionalities are grouped and placed to separate PMs) and rescheduling (*i.e.*, rescheduling of VMs from failed host to surviving hosts using live-migration). At storage or database level, fault tolerance is achieved by using replication and synchronization to ensure that a failure occurred at one device will not break the whole system.
- Availability: In OpenStack, high availability can be achieved through different setups depending on services types, stateless or stateful services. Stateless services can provide answer to a request without requiring further information of other services or historical data. OpenStack stateless services include nova-api and nova-scheduler. For these services, high availability is achieved by providing redundant instances and load balancing them. In the opposite, stateful services are ones that require other information to answer a request, which difficult the high availability achievement. These services, *e.g.*, database or storage, can be highly available by using replication, with the adequate synchronization between the main version and replicated versions in order to keep the system consistency [23].
- Security: OpenStack has a separated service (Identity service) which provides a central authentication management across the cloud operating system and users. The possibility to set up VPNs and firewalls is available.
- Compatibility: OpenStack is compatible with Amazon EC2 and Amazon S3 and thus client applications written for Amazon Web Services can be used with OpenStack with minimal porting effort [6]. In terms of hypervisors, OpenStack supports multiple hypervisors, *e.g.*, Xen, KVM, HyperV, VMWare, etc. Other hypervisors with existing standard drivers can also be interfaced with OpenStack through standard library, *e.g*, libvirt library.

3. CloudStack. CloudStack [4] is an open source software platform, written in Java, designed for development and management of cloud Infrastructure as a Service. It aggregates computing resources for building private, public or hybrid clouds. CloudStack brings together the "Stack" of features requested by companies and users, like data centers orchestration, management and administration of users and NaaS (Network as a Service).

The start of CloudStack was with Cloud.com in 2008. In May 2010, it was open source under GNU General Public License. Citrix bought CloudStack in July 2011, then in April 2012, Citrix donated CloudStack to Apache Software Foundation (ASF) where it was relicensed under Apache 2.0 and accepted as an incubation project. Since March 2013, CloudStack became a Top Level Project of Apache. Many companies are basing on CloudStack for building and managing their cloud infrastructures. Among these companies, there are: Nokia, Orange, Apple, Disney and many others.

The first major release of CloudStack since its graduation from the Apache Incubator is 4.1.0, major changes were realized on the platform with modifications in the codebase to make its development easier, Maven was used as a build tool, and for creating RPM/Debian packages a new structure was used. The second major release was 4.2, many features were born due to cooperations with different industries, including an integrated support of Cisco UCS compute chassis, SolidFire storage arrays, and the S3 storage protocol. Next CloudStack releases until the last one 4.5.1, mainly include new features, bug fixes and improvement of the interaction with different hypervisors.

3.1. General Architecture. In CloudStack, physical resources are organized and managed in a hierarchical structure. The lowest level contains the computational devices, *i.e.* hosts and primary storage. Different hosts accessing a shared storage form a cluster. Clusters combined by a layer 2 switch form a pod. Pods are grouped together with secondary storage by a layer 3 switch to form a zone. At the highest level, zones are grouped to create a region. All these resources are managed by a Management Server. In the following, we describe in details each component that forms the whole architecture.

3.1.1. The architectural levels. The architectural levels are the core of the CloudStack architecture. These levels are: the lowest level, clusters, pods, zones and regions.

A *host* represents a physical computational machine that contains local storage. The physical hosts are virtualized by hypervisors. CloudStack supports many hypervisors for VMs management such as Xen, KVM, vSphere, Hyper-V, VMWare, etc. as well as bare metal provisioning. All hosts within a cluster must be homogeneous in terms of the hypervisor, with the possibility of having heterogeneous hypervisors in different clusters.

Within a *cluster*, hosts are tied together into the same computational pool with the primary storage. In a cluster all hosts share the same IP subnet. The primary storage can be any kind of storage supported by the hypervisor and one cluster can have multiple primary storage devices.

A *pod* is a collection of different clusters linked with a layer 2 switch. Hosts in the same pod are in the same subnet. A pod is not visible to the end user.

A *zone* contains pods that are attached to the secondary storage using a layer 3 switch. Zones are visible to the end user and they can be private or public. Public zones are visible to all users in the cloud, while private zones are visible only to users from a particular domain. Zones main benefits are isolation and redundancy. Often, a zone corresponds to a data center; although if a data center is large enough, it can have multiple zones.

A *region* is the largest organizational unit in CloudStack. A region contains multiple zones distributed in geographic locations close to each other.

3.1.2. Management Server. A Management Server is used to manage all resources in cloud infrastructure through APIs or UI. One management server can support around 10K hosts and can be deployed either on a physical server or on a VM. In case of the existence of more than one management server, user interaction to either of them will return the same result. This ensures high availability of CloudStack.

A database is required for management servers to be persistent. In order to prevent single point of failure, we can have one primary database and several database replicas always synchronized with the primary copy.

3.1.3. Storage. In addition to the host local storage, CloudStack manages two main types of storage: primary storage and secondary storage.

- Primary storage: is a storage associated with a cluster or a zone, with the possibility of a single cluster to deploy multiple primary storages. This kind of storage is basically used to run VMs and to store application data. Since this storage interacts directly with applications deployed in VMs, it can be expensive in terms of I/O operations, reason why it is placed physically near to the hosts.
- Secondary storage: supports two different types, NFS and Object Storage. It is used to store ISO images, templates and snapshots.
  - ISO image: is used when user wants to create a VM.
  - Template: is the base operating system image that the user can choose when creating new instance. It may also include additional configuration information such as installed applications.
  - Snapshot: is used as backup for data recovery service. CloudStack supports two types of snapshot: individual snapshot and recurring snapshot. The former is one-time full snapshot, while the latter is either one-time full snapshot or incremental snapshot.

3.1.4. Networking. CloudStack supports the use of different physical networking devices (*e.g.* NetScaler, F5 BIG-IP, Juniper SRX, etc). In CloudStack, users have the ability to choose between two types of network scenarios: basic and advanced. The basic scenario is for an AWS-style networking. It provides a single network where guest isolation is done through the layer 3 switch. The advanced scenario is more flexible in defining guest networks [4]. For example, the administrator can create multiple networks to be used by the guests. CloudStack provides many networking services. Examples include:

- Isolation: CloudStack assures the isolation of networks, by allowing the access to the isolated network only by virtual machines of a single account.
- Load Balancing: to balance the traffic in the cloud, the user can create a rule to control and distribute the traffic and apply it to a group of VMs. Within the defined rule, user can choose a load balancing algorithm among the supported ones.
- VPN: to access a VM using a CloudStack account, users can create and configure VPNs. Each network has its own virtual router, so VPNs are not shared across different networks. Using VPN tunnels, hosts in different zones are allowed to access each other.
- Firewall: one host can access others in the same zone without passing through the firewall. Users can use external firewalls.

3.2. Properties. The main properties of CloudStack are [4]:

- Live migration: A live migration of running VMs between hosts is allowed in CloudStack through the Dashboard. Depending on the VMs hypervisor, migration conditions can be different. For example, live migration using KVM hypervisor will not support the use of local disk storage, and source and destination hosts have to be in the same cluster; while Xen and VMWare support local disk storage and allow to migrate between different clusters [3].
- Load balancing: a load balancer is an optional component of CloudStack that allows traffic distribution among different management servers [5]. In addition to creating rules and using load balancing algorithms, CloudStack offers the possibility to integrate with external load balancers such as Citrix NetScaler [27].
- Fault tolerance: in CloudStack, fault tolerance is achieved at different scales, *e.g.* management, database and host levels. In order to prevent failures of management server, the server can be deployed in multi-node configuration. If one management node fail, other nodes can be used without affecting the functioning cloud. Failures at database level are handled by using one or more replication of the database linked to the management server. For host's fail-over, CloudStack recovers the VM instances by taking the images from secondary storage and using application data in primary storage.
- Availability: CloudStack ensures high availability of the system by using multiple management server nodes which may be deployed with load balancers.
- Security: in addition to isolation using different accounts, VPNs and firewalls, CloudStack offers the isolation of traffic using the strategy of security groups. These are sets of VMs that filter the traffic on the basis of configuration rules. CloudStack provides a default security group with predefined rules, that can be modified if necessary.
- Compatibility: the pluggable architecture of CloudStack allows one cloud to support different hypervisor

implementations, including: Hyper-V, KVM, LXC, vSphere, Xenserver , Xen Project and also bare metal provisioning. Moreover, CloudStack is compatible with Amazon API and enables the integration of these two platforms.

- Scalability: CloudStack can manage thousands of servers distributed in different data centers and different locations, thanks to the management server capability (one management server node can manage a big pool of physical resources), and the possibility of using multiple management servers for reducing VMs downtime.
- API extensibility: The CloudStack APIs are very powerful and allow developers to create new command line tools and UIs, and to plug them into CloudStack architecture. If the developer wants to use a new hypervisor, new storage system or new networking service, he just needs to write a new plug-in in Java and integrate it.

4. Eucalyptus. Eucalyptus is a popular open source IaaS product, provided by Eucalyptus Systems [7]. It is used to implement, manage, and maintain private and hybrid clouds (but cannot build public clouds) with a main key design feature, which is Amazon Web Services (AWS) API compatibility. Many programming languages can be used to code Eucalyptus including: Java, C, Groovy, Shell, Perl, Python [7].

Originally it was designed at the University of California, Santa Barbara, as set of services that could emulate AWS on a different site apart from Amazon servers, with the aim of linking together AWS, supercomputer centers at National Science Foundation and several university sites [20]. It became a for-profit organization in 2009. In 2012, Eucalyptus started a partnership with AWS that allowed to develop more AWS-compatible environments, and to create hybrid clouds by facilitating movements of instances -created from stored Operating System Images- between an Eucalyptus private cloud and Amazon Elastic Compute Cloud (EC2). In September 2014 HP acquired Eucalyptus and lunch it under the Helion Eucalyptus name.

Beside its high AWS compatibility that allows running an application on AWS and Eucalyptus without any modifications, Eucalyptus is characterized by its high availability configuration, easy installation and simple user interface. Its main clients include: AppDynamics, Nokia, NASA, Puma and others [7].

Eucalyptus since its first releases contains its main feature which is compatibility with Amazon Web Services. Many announced releases were just maintenance releases and do not contain new features. Eucalyptus Cloud User Console was included in the release 3.2.0 which enabled self-service provisioning of different resources for cloud users. Eucalyptus 3.3.0 introduced many important features such as auto-Scaling, Load Balancing, CloudWatch and new VM Types, and this make it one of the most important releases of Eucalyptus. Many things were changed in release 4.0.0, including changes in the whole architecture and how components are installed and registered, changes in storage architecture, changes in networking mode and a new administrator roles were set. The current release 4.1.1 (11/5/2015) is a maintenance release for 4.1.0 in which a support for CloudFormation was add with new Management Console features and new instance status checks.

4.1. General Architecture. Eucalyptus has a highly modular, hierarchical and distributed architecture [22]. Users familiar with AWS do not find any difficulties with Eucalyptus because it replicates the same interaction tools and interfaces used in AWS, such as: euca2ools - the command-line tool - or the Eucalyptus User Console - a GUI based tool. Eucalyptus architecture is characterized by five main components: Cloud Controller, Cluster Controller, Storage Controller, Node Controller and Scalable Object Storage, and one optional component: VMware Broker. These components are grouped in three different logical levels: Cloud Level, Cluster Level and Node Level [7]. In the following sections we describe each logical level and the associated components.

4.1.1. Cloud Level. Contains two components: The Cloud Controller and the Scalable Object Storage.

The *Cloud Controller* (CLC) is a Java program that provides the interface to the cloud, and each cloud contains only one. CLC contains query interfaces and a EC2-compatible SOAP. It handles users requests and provides high level authentication, quota management, accounting and reporting. CLC does also meta-schedule and manages different cloud resources after collecting information provided by the Cluster Controller.

The *Scalable Object Storage* (SOS) is an Eucalyptus service that allows the use of external (open source or commercial) storage solutions. SOS is equivalent to AWS Simple Storage Service (S3). In case of small deployments Eucalyptus has a basic storage implementation called Walrus that has two main functionalities:

(i) storage of system files that could be accessible from different nodes (it may contain: VM images, volumes, snapshots, Linux kernel images, Root filesystem), and (ii) be used as Storage as a Service to store users data and applications.

4.1.2. Cluster Level. Each cluster is formed by a group of nodes linked with a LAN network. This level contains two main components: Cluster Controller and Storage Controller, and one optional: VMware Broker. Clusters are under subnets with a specific range of IP addresses, what compromises the flexibility and is a disadvantage of Eucalyptus.

The *Cluster Controller* (CC) is a C program that collects information, manages the execution of the virtual instances and virtual network and verifies the respect of Service Level Agreement. Multiple Cluster Controller could exist within one cloud. It works as an intermediate between the Cloud Controller, the Storage Controller and the Node Controller. CC is equivalent to AWS availability zone.

The *Storage Controller* (SC) is a Java program developed to have the same features as AWS Elastic Block Store (EBS). It manages the snapshots and volumes of a specific Cluster and controls the block-access network storage. The SC also communicates with different storage systems (NFS, iSCSI, SAN,...), with the Node Controller and the Cluster Controller.

The *VMware Broker* (VB) is an optional component available only in Eucalyptus version with VMware support. It provides an AWS compatible interface for VMware environments that allows the deployment of VMs on VMware infrastructure elements. VB directly manages the communication between the CC and the VMware hypervisors (ESX/ESXi), or it is possible that it passes through VMware vCenter.

4.1.3. Node Level. This level is composed by a single component, the Node Controller (NC), a C program that hosts VMs and their associated services, and manages the endpoint of the virtual network. NC interacts with the hypervisor and the hosted operating system to control VMs life-cycle, their creation and termination. It collects and sends information about VMs and their associated physical resources to the Cluster Controller to make high level decisions, like when to proceed with the load balancing.

4.2. Properties.

- Compatibility: it is the main feature of Eucalyptus. AWS APIs are built on top of Eucalyptus tools which simplifies intercommunication between both [25]. AWS APIs supported by Eucalyptus are: Elastic Compute Cloud (EC2), Elastic Block Storage (EBS), Amazon Machine Image (AMI), Simple Storage Service (S3), Identity and Access Management (IAM), Auto Scaling, Elastic Load Balancing, CloudWatch and CloudFormation.
- Live migration: one of the weak points of Eucalyptus is the absence of a live migration feature. Nevertheless many external approaches was proposed to add this feature, like the one described in [19].
- Load balancing and Fault tolerance: Eucalyptus does not contains an implicit load balancing mechanism for the low level of VMs, nor for the high level of user requests, but it could be achieved using Elastic Load Balancing of AWS. Elastic Load Balancing is a service that provides fault tolerance by distributing the incoming service requests and users traffic among different Eucalyptus instances. It automatically detects overloaded instances, and redirects the traffic to more available ones. Load balancers are the core of Elastic Load Balancing, they are special Eucalyptus instances created from specific Eucalyptus VMs images. If there is a problem in one of the instances or if it is removed due an internal error, the system stops rerouting traffic to that instance until it is restored or a new one is created. Load balancing could be managed within the same or among different clusters depending on the users need. Elastic Load Balancing contains also a health check system that routinely sends check requests to instances. It uses latency, RequestCount and HTTP response code counts to verify if the instance is responding in time to users requests.
- Scalability: as mentioned before, Eucalyptus is by nature distributed, what makes it highly scalable. There is also an auto scaling feature that allows to add and remove instances and VMs depending on traffic increase, the available resources and with respect to the Service Level Agreement. Using this feature developers can scale resources up or down depending on the need. There are three main component in Auto Scaling which are:
  - Auto Scaling group: is the main component of Auto Scaling, it defines for each user the minimum

and the maximum number of scaling instances, and the related parameters. In case the user did not select any specifications, it chooses the default parameters, which are equal to the minimum number of instances to use.
  - Launch configuration: it contains the information needed to make the scaling, including instance type, VM image ID, security groups, and many others.
  - Scaling plan (policy): it defines the manner of doing the auto scaling. It could be manually or automatically, responding to CloudWatch alarms.
- Cloud Watch: is an Eucalyptus service that collects raw data from different cloud resources (instances, elastic block store volumes, auto scaling instances and load balancers) and generates and records performance metrics. This allows users to make operational business decisions based on the historical records. Cloud Watch also configures alarms based on data from user filled metrics.
- Availability: Eucalyptus contains high availability as a feature since version 3. If an individual node or even a rack fails, Eucalyptus put other nodes into use immediately or moves to another rack in case of rack failure. This is achieved through a service that is running concurrently on physical machines which is "hot spare". The information failure is quickly diffused internally, without any signs to the users, and the failure is recovered with respect to SLA (service level agreement). In [30] an evaluation tool was proposed for testing availability in IaaS platforms. This tool was tested on Eucalyptus by faults injections in an Eucalyptus cloud testbed. This paper shows that software repairs were more often than hardware repairs.
- Security: Eucalyptus manages the access to the cloud using policies related to users, groups and accounts. A group is a set of users within an account, which have the authorization to access to a specified pool of resources. A user can belong to different groups. Security groups are also used by defining firewalls that should be applied to the set of VMs within a group. The security policies could be managed by users using the command line Euca2ools.

5. OpenNebula. OpenNebula [2] is an open source cloud platform developed by Universidad Complutense of Madrid UCM (Under the Apache 2.0 License) that delivers a simple, but feature rich, solution to build enterprise clouds and virtualized data centers. OpenNebula provides a complete toolkit to centrally manage heterogeneous virtual infrastructure, which is compatible with conventional hypervisors: VMware, Xen, KVM. It operates as a scheduler of storage layers, network, supervision and security. It is an appropriate solution for the conversion of a virtual infrastructure to IaaS platform. The centralized orchestration of hybrid environments is the heart of the tool. OpenNebula also utilizes Cloud Computing Interface (OCCI) and support to Amazon Elastic Cloud Compute (EC2) in order to expand the resources connected to it in order to form a hybrid cloud [2].

OpenNebula project started in 2005, has delivered its first version in 2008 and remains active since. Many releases have achieved today significant functional changes on the support of the storage nodes, high availability environments and ergonomics of the administrative interfaces. OpenNebula has also a wide user base that includes leading companies in banking, technology, telecommunications, and research and supercomputing centers. At present, it has more than 4000 downloads per month and many research institutes and enterprises use it to build their own cloud.

One major release of OpenNebula is launched approximately every year, with two or three minor releases to each version. The first major release was in July 2008; it supported Xen and KVM virtualization platforms to provide efficient resource management and fault tolerant design. OpenNebula 2.0 (25/10/2010) brought a significant amount of changes and new features, including: the image repository, MySQL support, authorization and authentication drivers. The next major release of OpenNebula was 3.0 (3/10/2011) and introduced many new components and features for the core and libraries, providing support to: groups users management, flexible access control list system, DB versioning and schema. Currently, the stable version of OpenNebula is 4.12.1 (8/04/2015) and presents several improvements in resource management with virtual data-centers and the exclusion of the term resource provider. Networking has been vastly improved in 4.12 (03/11/2015), with the addition of security groups, allowing administrators to define the firewall rules and apply them to the Virtual Machines.

5.1. General Architecture. A key feature of OpenNebula architecture is its highly modular design and flexibility, which facilitates integration with any virtualization platform and third-party component in the cloud ecosystem. The main components of OpenNebula architecture are: the Driver, the Core and the Tools.

The *Driver* is responsible for direct communication with the underlying operating system and for the encapsulation of the underlying infrastructure as an abstract service (e.g. virtualization hypervisor, transfer mechanisms or information services). It is designed to plug-in different virtualization, storage and monitoring technologies and cloud services into the core. These pluggable drivers are responsible for the creation, startup and shutdown of virtual machines, allocating storage for VMs and monitoring the operational status of physical machines and VMs. The roles of each service are:

- The transfer driver manages the VMs disk images on different kind of storage systems, a shared one: Network File System (NSF) or Internet Small Computer System Interface (iSCSI), or a non-shared one, such as a simple copy over Secure Shell (SSH).
- The VM driver is considered a set of hypervisor-specific drivers used to manage VMs instances on different hosts.
- The information driver is also considered a set of hypervisor-specific drivers used to monitor and retrieve the current status hosts and VMs instances through SSH.

The *Core* reflects a centralized layer that controls and monitors VM full life cycles, virtual networks (VN), storage and hosts. These components are implemented in this layer by invoking a suitable driver. The features of such component are: *(i)* VM manager allocates resources required by VMs to operate, implementing VMs deployment policies; *(ii)* VN manager interconnects VMs, and generates MAC and IP address for a VM; *(iii)* host manager manages a VM storage and allocates a VM disk; *(iv)* SQL Pool is a database (SQLite or MySQL) that stores configuration data and current status of hosts and VMs instances; and *(v)* request manager (XML-RPC) accesses the application programming interface directly.

The *Tools* contains tools distributed with OpenNebula. First, it includes Scheduler that manages the functionality provided by the core layer. Scheduler component makes VM placement decisions. These VMs are deployed on host nodes following specific user requirements and resource-aware policies, such as packing, striping, or load-aware. Secondly, Command line interface-CLI and Libvirt API [1], an open interface for VM management and communicating with users. Additionally, third party tools that can be easily created using the XML-RPC interface or the OpenNebula Client API. External users are capable of sharing these functionalities through a cloud interface which is provided by the Tools layer.

5.2. Properties.

- Live migration: is one of the advantages of OpenNebula. It is supported through shared storage, but it could demand a high-performance SAN (Storage Area Network) [18]. OpenNebula uses the libvirt TCP protocol to provide migration capabilities.
- Load balancing: is provided across NGINX [34], which is an open-source, high-performance web server. It is capable of handling large numbers of concurrent connections and represents a centralized manager to balance the workload. In order to distribute efficiently the I/O of the VMs across different disks, LUNs or several storage back-ends, OpenNebula is able to define multiple system datastores per cluster. Scheduling algorithms (used by load balancers) take into account disk requirements of a particular VM, so OpenNebula is able to pick the best execution host based on capacity and storage metrics [9].
- Fault tolerance: OpenNebula provides VM migration for those not running VMs. However, OpenNebula can only detect problems on VM level. If a service or an application corrupts unexpectedly, simply rebooting the VM on another node may break the continuity of the service and result in loss. This service is maintained by database back-end (registers VM information) to store host and VM information.
- High availability: OpenNebula delivers the availability required by most applications running in VMs. OpenNebula ensures high availability of the system by using multiple persistent databases back-end in a pools cluster of hosts that share datastores and VNs. It provides information in order to prepare for failures in the VMs or physical nodes, and recover from them. These failures are categorized depending on whether they come from the physical infrastructure (Host failures) or from the virtualized infrastructure (VM crashes). In both scenarios, OpenNebula provides a cost-effective fail-over solution to minimize downtime from server and OS failures, and supports high availability configurations [15].

- Security: OpenNebula takes many measures to ensure the security. The infrastructure administrator manages a secure and efficient Users and Groups Subsystem for pluggable authentication and authorization based on passwords, SSH and RSA key pairs, X.509 certificates or LDAP. OpenNebula also uses Firewall to configure the VMs, in order to shutdown TCP and UDP ports, filter some unwanted packets and defines a policy for ICMP connections. In addition, OpenNebula uses ACL (Access Control List) which is a collection of permit and deny conditions (ACL rules), allowing different role management with fine grain permission granting over any resource managed by OpenNebula, support for isolation at different levels. Since version 4.4 OpenNebula has special authentication mechanisms for SunStone (OpenNebula GUI) and the Cloud Services (EC2 and OCCI).
- Compatibility: OpenNebula can be deployed to existing infrastructures and integrated with various cloud services and multi-platform. OpenNebula currently includes an EC2 driver, which can submit requests to Amazon EC2 and Eucalyptus, as well as an ElasticHosts driver. OpenNebula supports different access interfaces including REST-based interfaces (e.g., EC2-Query API), OGF OCCI service interfaces, the OpenNebula Cloud API (OCA), and APIs for native drivers, for example, to connect with AWS.
- Scalability: OpenNebula has been tested in the management of medium scale infrastructures with hundreds of servers and VMs [15]. By a cloud federation, OpenNebula provides scalability, isolation, and multiple-site support to interface with external clouds. This allows complementing the local infrastructure with computing capacity from public clouds to meet peak demands. Thus, a single access point and centralized management system can be used to control multiple deployment on OpenNebula. In OpenNebula highly scalability can be achieved through database back-end with support for MySQL and SQLite, and virtualization drivers can be adjusted to achieve maximum scalability.
- Flexibility and extensibility: OpenNebula provides extension and integration to fit into any existing data center, and flexible architecture, interfaces and components, allowing its integration with any product or tool. OpenNebula offers different means to easily extend and adjust behavior of the cloud management instance to the requirements of the environment and use cases, *e.g.* new drivers can be easily written in any language for the main subsystems to easily leverage existing IT infrastructure and system management product [9].

6. Nimbus.

6. Nimbus. Nimbus is an open source solution (licensed under the terms of the Apache License) for using cloud computing in the context of scientific applications. Although the focus on the scientific community, Nimbus approaches three goals targeting three different communities: (1) Enable resource owners to provide their resources as an infrastructure cloud; (2) Enable cloud users to access infrastructure cloud resources more easily, and (3) Enable scientists and developers to extend and experiment with both sets of capabilities. These goals are related with the architecture of Nimbus. Its main architecture can be divided in two components, each one responsible for attending one of the goals. The first goal is realized by the Nimbus Infrastructure and the second by the Nimbus Platform. The third goal is realized by the strong support of open source development practices via modular, extensible code and engagement with open source developers [8].

Released in 2005, to solution is kept on GitHub since 2009 and its updates are revised by an international researchers committee. Nevertheless, recently, Nimbus is missing regular updates. The last release is 2.10.1 (release date: 27/02/2013) and the last github update is the cloud client version 022 (release date: 27/10/2013). Although some new features have been introduced, like the multi-cloud VM image generator for FutureGrid users (release date: 04/06/2014) and some additions on the Phantom component (release date: 15/07/2014), the focus of the Nimbus team seems to be more on collaborating with another research groups than on developing the Nimbus.

6.1. General Architecture. This section presents further details on the components Nimbus Platform and Nimbus Infrastructure.

6.1.1. Nimbus Platform. This is an integrated set of open source tools that allows users to easily leverage Infrastructure-as-a-Service (IaaS) cloud computing systems. This includes application instantiation, configuration, monitoring, and repair [8]. The Nimbus platform is divided in three modules: the context broker, the elastic scaling tools and the deployment coordination.

The *context broker* is a service that allows clients to coordinate large virtual cluster launches automatically and repeatably. The context broker requires that each VM runs a lightweight script at boot time called the context agent. This context agent depends only on Python and on the ubiquitous curl program, that securely contacts the context broker using a secret key. To enforce security the context broker uses contextualization, *e.g.*, the key is created on the fly and seeded inside the instance. This agent gets information concerning the cluster from the context broker and then causes last minute changes inside the image to adapt to the environment [8].

The *elastic scaling tools* on Nimbus are called EPU. The EPU system is used with IaaS systems to control highly available services. On these kind of services any failures are compensated with replacements. EPU is also useful with services that can be configured to be elastic; it responds to monitoring signals with adjustments of the amount of instances that composes a service. If the service cannot handle instances being added and dropped on the fly (many services have static node-number configurations), EPU still provides automatic launch, monitoring, and failure replacement capabilities [8].

6.1.2. Nimbus Infrastructure. This is a set of tools that provides the IaaS at the Nimbus cloud computing solution. The Nimbus infrastructure is divided in Workspace service and Cumulus.

The *Nimbus workspace service* is a standalone site VM manager that can be invoked by different remote protocol front-ends [8]. The workspace service is web services based and provides security with the GSI authentication and authorization. Currently, Nimbus supports two front-ends: Amazon EC2 and WSRF. The structure of the workspace service is composed by three modules: workspace control, workspace resource manager and workspace pilot.

- Workspace control: controls VM instances, manages and reconstructs images, integrates a VM to the network and assigns IP and MAC addresses. The workspace control tools operate with the Xen hypervisor and can also operate with KVM. Implemented in Python in order to be portable and easy to install [35, 8].
- Workspace resource manager: is an open source solution to manage different VMs, but can be replaced by other technologies such as OpenNebula [35, 8].
- Workspace pilot: is responsible for providing virtualization with few changes in cluster operation. This component handles signals and integrates administration tools [35, 8].

The *Nimbus Cumulus* is an open source implementation of the Amazon S3 REST API. It provides an implementation of a quota-based storage cloud. In order to boot an image on a given Nimbus cloud, that image must first be put into that same clouds Cumulus repository, although advanced use cases can by pass this restriction. In practice, it is used as the Nimbus repository solution but can also be installed standalone. Cumulus is designed for scalability and allows providers to configure multiple storage cloud implementations [8].

6.2. Properties.

- Live migration: Nimbus has no in built support to live migration. It is possible to migrate virtual machines only at hypervisor interface level [26]. The Nimbus team recently participated in a research that addresses network contention between the migration traffic and the Virtual Machine application traffic for the live migration of co-located Virtual Machines, and the live migration support is on the map of next improvements of the solution [16].
- Load balancing: At VM level, Nimbus features a system of Nagios plugins that can give information on the status and availability of the Nimbus head node and worker nodes, including changes of the virtual machines running on the worker node [24]. Specifically from the cloud provider's point of view Nimbus does the back-filling of partially used physical nodes, allowing also preemptable virtual machines. On the platform level, the cloudInit.d tool provides management to virtual machines deployed and allows compensation of stressed workloads based on policies and sensor information. Nimbus also provides tools to handle capacity allocation and capacity overflow. For example, the attribution of variable lease limits to different users - as a means of scheduling - is standard within Nimbus. In addition, the idea of allowing EC2 or another cloud the ability to pick up excess demand is heavily researched with Nimbus [28].
- Fault tolerance: Nimbus presents fault tolerance only at storage level, through the integration of Nimbus Storage Service with Globus GridFTP [10]. GridFTP - an extension of the standard File Transfer Protocol (FTP) for use with Grid computing - provides a fault tolerant implementation of FTP, to

handle network unavailability and server problems. Moreover, transfers can be automatically restarted if a problem occurs [31].

- Availability: On the platform point of view, the EPU provides high accessible services to the user. On the infrastructure point of view, Nimbus provides high-available services through the hosted service Phantom. The Nimbus Phantom leverages on-demand resources provided by infrastructure clouds and allows users to scale VMs that are running on the many clouds of FutureSystem as well as Amazon EC2. The Phantom can be extended through decision engines, components that determine the behavior of the service. Phantom is freely available on the FutureSystem infrastucture and is provided as a highly available service itself.
- Security: Nimbus provide GSI authentication and authorization - through PKI credentials, grid proxies, VOMS, Shibboleth (via GridShib) and custom PDPs. It also guarantees secure access to VMs via EC2 key generation. In addition, Nimbus allows images and image data validation.
- Compatibility: Nimbus can be integrated with various cloud services and multi-platform mainly through web services. Nimbus Infrastructure is EC2/S3-compatible, with SOAP and Query front-ends. It is possible to upload VM images to Cumulus with the Python library Boto, or with s3cmd. It is also possible to use EC2 spot instances.

7. Cloud Solution Comparisons. This section provides a comparison between the analyzed cloud solutions, aiming at three different levels: *(i)* general comparison aimed at providing high level analysis in terms of model, policy and architecture, *(ii)* functional comparison, whose goal is to compare supported functionalities, and *(iii)* property comparison, which considers cloud properties implemented in these platforms.

7.1. General Comparison. The general comparison is provided in Table 7.1, considering general aspects: licensing, commercial model, cloud model compatibility, easiness of installation, architecture and adopters.

Table 7.1
General comparison

Property	OpenStack	CloudStack	OpenNebula	Eucalyptus	Nimbus
Open Source License	Apache 2.0	Apache 2.0	Apache 2.0	Linux Open-Source	Apache 2.0
Commercial model	Free	Free	Free	Free, GPLv3 (only), with proprietary relicensing	Free
Compatibility with	Private, public and hybrid clouds	Private, public and hybrid clouds	Private, public and hybrid clouds	Private and hybrid clouds	Private, public and hybrid clouds
Installation Effort	Difficult (many choices, not enough automation)	Medium (Few parts to install)	Easy (process based package installers)	Difficult (different configuration possibilities)	Easy (no root account required)
Architecture	Fragmented into many pieces	Monolithic controller	Modular (third-party component)	Five part controller and AWS	Lightweight components (IaaS service and VMM node)
Large organizations adopters	Yahoo, IBM, VMWare, Rackspace, Redhat, Intel, HP, etc.	Nokia, Orange, Apple, Citrix, Huawei, TomTom, Tata, etc.	CERN, Cloud-Weavers, IBM, Hexafrid	UEC, NASA, Sony, HP, Cloudera, Puma, USDA, FDA	Brookhaven National Labs, Cumulus project

As it can be seen, the five frameworks are generally equal with respect to commercial model, licensing policy and cloud models. Each of them has been adopted by large organizations. Nevertheless, a big difference is spotted from the architecture viewpoint. While OpenStack is fragmented into modules, CloudStack has a monolithic central controller, OpenNebula has three main components, Eucalyptus has five parts controllers with AWS, and Nimbus have lightweight based components. This difference is explained by the open philosophy of Open Stack and OpenNebula which tries to avoid technology lock-ins and provides high degree of flexibility, extension and availability. Nimbus is also relatively easy to install and deploy comparing to others solutions. The decentralized design and different configuration possibilities of Eucalyptus make it difficult to configure and install. Like Eucalyptus, OpenStack are also characterized by increasing complexity of installation and configuration.

7.2. Functional Comparison. The functional comparison looks at the offered functionalities or technical aspects of the five presented solutions, as described in Table 7.2. Most popular hypervisors such as: Xen and KVM are supported by all the platforms, but for example VMware is not available on Nimbus. We mention here that OpenStack and CloudStack have the largest number of supported hypervisors, even though there are ways to interface with non-supported ones. Administration feature is the interface available to interact with these platforms. All solutions present Web interfaces (Web UI) and command line interfaces (CLI). Also user management is provided in all this five platforms.

Table 7.2
Functional comparison between the open source Cloud solutions

Functionality	OpenStack	CloudStack	Eucalyptus	OpenNebula	Nimbus
Supported hypervisors	Xen, KVM, HyperV, VMWare, LXC, vSphere	Xen, KVM, HyperV, VMWare, LXC, vSphere	Xen, KVM, VMware	Xen, KVM, VMware, vCenter	Xen, KVM
Administration	Web UI, CLI	Web UI, CLI	Web UI, CLI	Web UI, CLI	Web UI, CLI
User management	yes	yes	yes	yes	yes

Other important functional aspect of open source cloud solutions is the activeness of its community. A solution that is always seeing new releases is constantly evolving. An active community, with well documented wiki, good bug reporting and fixing system and active users support are fundamental features for the success of an open source solution. At this topic OpenStack remains the largest and most active open source cloud computing project [21]. Nevertheless, CloudStack and Eucalyptus are growing and have an important participation in the market, while OpenNebula and Nimbus are less active. In this point, OpenStack has a bigger and more active community, followed by CloudStack, Eucalyptus, OpenNebula and finally Nimbus.

The number of releases of a platform also indicates the constant evolution of a tool. OpenStack has a strict policy of regular updates, with two big releases per year and some minor releases in each version. For Eucalyptus the last major release is Version 4.1.1 (release date: 11-05-2015) and the one before was Version 4.0.2 (release date: 20-10-2014). On CloudStack the last major release was 4.5.1 (release date 03/06/2015). For OpenNebula the last major release is old, the 4.4 Beta (release date: 07-11-2013), but 13 minor releases were announced since that time, the last one is 4.12.1 (release date: 08-04-2015). The Nimbus last major release was 2.10.1 (release date: 27/02/2013) and a minor update on cloud client (release date: 2/10/2013). From this we can conclude that OpenStack, Eucalyptus and CloudStack are the most active and evolving technologies. OpenNebula also evolves, but in a slower pace, meanwhile Nimbus gives signs of possible discontinuity.

Regarding benchmark tests, as far as we know, there is no study that have made performance tests to compare this five platforms, but there are several papers comparing two or three of them. In [39] the authors performed the performance evaluation of CloudStack and OpenStack using a mutual hypervisor and under a set of defined criteria. This study showed the effect of varying resources performances (processor and RAM, hard disk size) on deployment and deletion time. The results showed that performance of OpenStack supersedes that of CloudStack. The authors in [40] compared performances of VMs for CloudStack and Eucalyptus in terms of CPU utilization, memory bandwidth, disk I/O access speed, and network performance using different benchmarks. Mainly the results shown that CloudStack performed better than Eucalyptus in most of tests.

Other benchmarks could be found in [41] where the authors evaluated performances of Nimbus, OpenNebula and OpenStack, for High Performance Computing according to HPC Challenge (HPCC) benchmark suite. The results showed that OpenStack had the best performance for HPC.

7.3. Properties Comparison. Properties comparison gives deeper insights into the five platforms, considering some important properties that an IaaS has to provide. The comparison is given in Table 7.3.

Table 7.3
Properties comparison between the open source cloud solutions.

Property	OpenStack	CloudStack	Nimbus	Eucalyptus	OpenNebula
Live migration	Yes	Yes	No	No	Yes
Load balancing	Yes	Yes	Yes	Yes	Yes
Fault tolerance	VM scheduling, replication	VM scheduling, replication	Through Globus GridFTP	Through AWS Elastic Load Balancing	VM scheduling, replication
High Availability	Yes	Yes	Yes	Yes	Yes
Security	VPNs, firewall, user authentication, others	VPNs, firewall, user management, others	user authentication	group and users policies	user authentication
Compatibility	Amazon EC2, Amazon S3	Amazon EC2, Amazon S3	Amazon EC2, Amazon S3, WSRF	Amazon EC2, Amazon S3	All Amazon Interfaces

Live Migration may be approached in two ways: shared storage based live migration, and block live migration. OpenNebula and OpenStack offer both possibilities. CloudStack also offers both possibilities, but the conditions change according with the user hypervisor. Eucalyptus and Nimbus offers no integrated live migration system. This way, if the live migration is sensitive to the application, OpenNebula and OpenStack are more adequate solutions.

Load balancing can be considered on the VM level or on the host level. Load balancing at host level is implemented in OpenStack through live migration, which is the same in CloudStack; Nimbus does the back-filling of partially used physical nodes, allowing also preemptable virtual machines. All the solutions approach VM level load balancing through the establishment of a plug-in architecture. OpenNebula can be coupled with NGINX, Eucalyptus with the ELB of AWS and Nimbus with Nagios plugins. Similarly OpenStack and Cloudstack present flexible plug-in architecture on network component. In resume, at host level, live migration is used to provide load balancing. At VM level, the cloud solutions trust on plugins to provide load balancing. Also, the automatic leasing of resources seems to be a tendency. For example, allowing EC2 or another cloud the ability to pick up excess demand is heavily researched with Nimbus [28].

Fault tolerance mechanisms exists on VM or on storage/database levels. At VM level, fault tolerance is approached under the policies to schedule VM placement or services replication. OpenNebula comes with a match making scheduler - that implements the Rank Scheduling Policy - and the quote management system, that ensure that any user gets a adequate quantity of resources. OpenStack has in-built scheduling algorithms (group scheduling and rescheduling) and newer ones can be implemented by the user. Nimbus has no already implemented fault tolerance system, but it can provide through Globus GridFTP. At storage or database level, fault tolerance is achieved by using replication and synchronization to ensure that a failure occurred at one device will not break the whole system. Eucalyptus has no in-built mechanism for fault tolerance, but can provide it at storage level, through the AWS Elastic Load Balancing.

High availability is approached in all platforms by means of using redundant service instances and load balancing to distribute workloads among those instances. In the case of the replication of service instances, some synchronization technique has to be considered.

Security is provided on different levels by each cloud solution. Centralized in-built user authentication is provided on Nimbus, OpenStack and OpenNebula. The establishment of security policies for users, groups and accounts is possible on CloudStack, Eucalyptus and OpenNebula. On OpenStack is also possible to extend the security of the cloud through the addition of plugins.

Compatibility refers to the capability of the cloud solution to integrate with other tools and cloud solution. In this topic, the Amazon Web Services are a common place on the solutions, thanks to its dominance on commercial public clouds. All open source analyzed solutions, in different levels, present some integration with AWS services, being Amazon EC2 and S3 the most popular.

7.4. Resume. In general lines, Eucalyptus offers a flexible solution for users that want privacy in specific modules while keep managing their clouds with AWS. OpenNebula is for someone interested in the internal technical details of the cloud, but also seems to be a good solution for someone that wants to build up a cloud quickly using just a few machines. OpenStack and CloudStack have the largest developing community, but have opposite approaches, since OpenStack has as modularized architecture, while CloudStack has a monolithic centralized one. Nimbus is easy to deploy and is suitable for inexperienced users willing to have a first contact with cloud platforms or to scientific investigations, although the lack of recent releases and an active community.

8. Conclusions. We have presented the up-to-date architecture of five prominent open source cloud platforms, looking into details of the provided functionalities and their properties. The analyzed platforms are under continuous development. Therefore their documentation and technical reviews are often updated and have to be regularly checked. We argue that there is no best solution to any general case, but there are tools more adapted to specific audiences. The comparison was carried out from the user perspective, considering the properties that a user needs to know when choosing a IaaS cloud solution.

Acknowledgment. Work funded by the European Commission under the Erasmus Mundus Green IT project (Green IT for the benefit of civil society, 3772227-1-2012-ES-ERA MUNDUS-EMA21, Grant Agreement n 2012-2625/001-001-EMA2). And CAPES, Coordenacao de Aperfeicoamento de Pessoal de Nivel Superior - Brasil.

REFERENCES

[1] *Libvirt home page, http:// libvirt.org.*

[2] *OpenNebula 3 Cloud Computing*, 2012.

[3] *Cloudstack live migration, http://cloudstack.apache.org/docs/en-us/apache_cloudstack/4.1.0/html/admin_guide/manual-live-migration.html*, 2014.

[4] *Cloudstack website, https://cloudstack.apache.org/docs/*, 2014.

[5] *Management server load balancing, https://cloudstack.apache.org/docs/en-us/apache_cloudstack/4.0.2/html/installation_guide/management-server-lb.html*, 2014.

[6] *Openstack history, https://www.openstack.org/*, 2014.

[7] *Eucalyptus website, https://www.eucalyptus.com/*, 2015.

[8] *Nimbus website, http://www.nimbusproject.org/*, 2015.

[9] *Opennebula home page, http://opennebula.org*, 2015.

[10] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, *Gridftp: Protocol extensions to ftp for the grid*, Global Grid ForumGFD-RP, 20 (2003).

[11] C. E. Amrani, K. B. Filali, K. B. Ahmed, A. T. Diallo, S. Telolahy, and T. El-Ghazawi, *A compartive study of cloud computing middleware*, in Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), IEEE Computer Society, 2012, pp. 690–693.

[12] A. Barkat, A. D. dos Santos, and T. T. N. Ho, *Open stack and cloud stack: Open source solutions for building public and private clouds*, in Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2014 16th International Symposium on, IEEE, 2014, pp. 429–436.

[13] S. A. Baset, *Open source cloud technologies*, in Proceedings of the Third ACM Symposium on Cloud Computing, (2012).

[14] C. V. Blanco, *The opennebula virtual infrastructure engine*, Distributed Systems Architecture Research Group, Universidad Complutense de Madrid, (2009).

[15] G. Chen, H. Jin, D. Zou, B. B. Zhou, W. Qiang, and G. Hu, *Shelp: Automatic self-healing for multiple application instances in a virtual machine environment.*, in CLUSTER, 2010, pp. 97–106.

[16] U. Deshpande and K. Keahey, *Traffic-sensitive live migration of virtual machines.*

[17] P. T. Endo, G. E. Gonçalves, J. Kelner, and D. Sadok, *A survey on open-source cloud computing solutions*, in Brazilian Symposium on Computer Networks and Distributed Systems, 2010.

[18] J. F., *The opennebula engine for on-demand resource provisioning*, in HEPiX Spring.

[19] N. Karkare, *A survey on the live migration of virtual machines.*

[20] R. Milojicic, D. Wolski, *Eucalyptus: Delivering a private cloud*, Computer, (2011).

[21] networkworld website, URL: http://www.networkworld.com/article/2166407/cloud-computing/stack-wars–openstack-v–cloudstack-v–eucalyptus.html, July 2014.

[22] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, *The eucalyptus open-source cloud-computing system*, in Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on, IEEE, 2009, pp. 124–131.

[23] OpenStack, *Introduction to openstack high availability*, in http://docs.openstack.org/high-availability-guide/content/stateless-vs-stateful.html, OpenStack.

[24] H. A. Patel and A. D. Meniya, *A survey on commercial and open source cloud monitoring*, International Journal of Science and Modern Engineering (IJISME), ISSN, (2013), pp. 2319–6386.

[25] S. G. Rakesh Kumar, *Open source infrastructure for cloud computing platform using eucalyptus.*

[26] P. Riteau, *Building dynamic computing infrastructures over distributed clouds*, in Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on, IEEE, 2011, pp. 127–130.

[27] P. N. Sabharwal, *Integrating netscaler with cloudstack*, in Apache CloudStack Cloud Computing, Packt Publishing.

[28] P. Sempolinski and D. Thain, *A comparison and critique of eucalyptus, opennebula and nimbus*, in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, Ieee, 2010, pp. 417–426.

[29] O. C. Software, *Chapter 1. introduction to openstack*, in http://docs.openstack.org/training-guides/content/module001-ch001-intro-text.html, OpenStack.

[30] D. Souza, R. Matos, J. Araujo, V. Alves, and P. Maciel, *Eucabomber: Experimental evaluation of availability in eucalyptus private clouds*, in Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on, IEEE, 2013, pp. 4080–4085.

[31] T. Viet-DINH, *Cloud data management*, ENS de Cachan, IFSIC, IRISA, KerData Project-Team, (2010), pp. 1–5.

[32] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, *Qualitative comparison of multiple cloud frameworks (2012).*

[33] I. Voras, B. Mihaljevic, M. Orlic, M. Pletikosa, M. Zagar, T. Pavic, K. Zimmer, I. Cavrak, V. Paunovic, I. Bosnic, et al., *Evaluating open-source cloud computing solutions*, in MIPRO, 2011 Proceedings of the 34th International Convention, IEEE, 2011, pp. 209–214.

[34] W. Zeng, J. Zhao, and M. Liu, *Several public commercial clouds and open source cloud computing software*, in Computer Science & Education (ICCSE), 2012 7th International Conference on, IEEE, 2012, pp. 1130–1133.

[35] P. T. Endo, G. E. Gonçalves, J. Kelner, and D. Sadok, "A survey on open-source cloud computing solutions," in *Brazilian Symposium on Computer Networks and Distributed Systems*, 2010.

[36] D. Petcu and M. Rak, "Open-source cloudware support for the portability of applications using cloud infrastructure services," in *Cloud Computing.* Springer, 2013, pp. 323–341.

[37] T. Cordeiro, D. Damalio, N. Pereira, P. Endo, A. Palhares, G. Gonçalves, D. Sadok, J. Kelner, B. Melander, V. Souza *et al.*, "Open source cloud computing platforms," in *Grid and Cooperative Computing (GCC), 2010 9th International Conference on.* IEEE, 2010, pp. 366–371.

[38] I. Voras, B. Mihaljević, and M. Orlić, "Criteria for evaluation of open source cloud computing solutions," in *Information Technology Interfaces (ITI), Proceedings of the ITI 2011 33rd International Conference on.* IEEE, 2011, pp. 137–142.

[39] A. Paradowski, L. Liu, and B. Yuan, "Benchmarking the performance of openstack and cloudstack," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on.* IEEE, 2014, pp. 405–412.

[40] A. A. A. M. Mumtaz M.Ali AL-Mukhtar, "Performance evaluation of private clouds eucalyptus versus cloudstack."

[41] C. Li, J. Xie, and X. Zhang, "Performance evaluation based on open source cloud platforms for high performance computing," in *Intelligent Networks and Intelligent Systems (ICINIS), 2013 6th International Conference on.* IEEE, 2013, pp. 90–94.

Edited by: Dana Petcu
Received: May 14, 2015
Accepted: Jun 24, 2015