Chapter 13
Development of Cloud Applications in Hybrid Clouds with Support for Multi-scheduling

Lucio Agostinho Rocha

Abstract Development of cloud applications must consider many aspects inherent in the distributed nature of clouds, mainly those related to elasticity, high access level to computational resources, multi-tenant behavior, transparency, pay-per-use model, and resource scalability. In addition, portability is a key feature that must be present in any development framework to allow extensions and simplify resource sharing by standardized interfaces. Open source approaches can be used, but the model must be composed of independent parts to optimize the availability of active components in the infrastructure. Hybrid cloud models are interesting because widely acceptable solutions can be developed without "reinventing the wheel." Private clouds are more suitable for keeping restricted data or supporting services of small enterprises or institutions. However, their infrastructure must offer alternatives to provide services outside their own domain. In this context, a private cloud can use frameworks of public clouds and aggregate services to support the development of new applications. This generally occurs in PaaS models, where the platform offers pre-configured tools to interact with services of other domains. Security issues must also be considered at all stages of development, as most of the communication takes place among services located in different domains, linked by Internet connections. Solutions such as OpenID guarantee that public cloud services are used for the purpose of authentication, but additional security features in the source domain must be assured. In this chapter, a development framework is presented to guide the development of widely acceptable cloud applications, following standardized open source solutions. This framework, originally developed for a robotic environment, can be extended to support other cloud environments. The study presents

L.A. Rocha (✉)
Department of Computer Engineering and Industrial Automation (DCA)
at the School of Electrical and Computer Engineering (FEEC),
State University of Campinas, São Paulo, Brazil
e-mail: l089278@dac.unicamp.br

aspects related to multi-scheduling of virtual machines and suggests how virtualized applications can be developed with different methodologies, such as dynamic IP, Web service with SOAP communication, MapReduce approach, and OCCI-based infrastructure.

Keywords Cloud computing • Hybrid cloud • OCCI • OpenID • Cloud framework • MapReduce • Virtualization

13.1 Introduction

The development of distributed cloud architectures deals with issues of scalability, elasticity over demand, broad network access, usage measurement, security aspects such as authorization and authentication, and many other concepts related to multi-tenant services in order to serve a high number of concurrent users over the Internet. The nature of a distributed cloud has implications about how the offered services are organized over different administrative domains. In order to extend the Service-Level Agreement (SLA) to thousands of users, the support architecture must have interfaces compatible with other cloud providers.

This work presents a cloud framework directed to the requirements of portability, respecting the Open Grid Forum (OGF) and Open Cloud Computing Interface (OCCI) patterns [1]. The framework has kernel components that guide the extension of the whole system. Also contemplated are the methodology, architecture, and wrapper of open source APIs, such as OpenID [2], to allow aggregation of other cloud services to the system. We discuss how other cloud technologies model their own structures. Our goal is to illustrate mechanisms to integrate private and public clouds in a hybrid model.

The above-mentioned concepts have been used to develop a real cloud laboratory offering different Linux operating systems as services. Unlike Amazon EC2 [3] or Windows Azure [4] cloud environments, in this cloud architecture, Linux systems can be used to interact with robotic resources accessible only inside the laboratory. In addition, this architecture allows the inclusion by the user of compatible virtual machines into the system. This system is unique in that it deals with network issues only during the period reserved for robotic experiments. The framework also supports multiple scheduling approaches, that is, multi-scheduling.

This framework was designed according to the Layered Design Pattern, a well-defined standard where lower levels provide services to higher ones. Each level is defined in such a way as to allow development independently from the others, according to interfaces compatible with open patterns such as OCCI.

SSL and X.509 digital certificates guarantee the security of Internet access from outside the institution. The main goal of this security infrastructure is to reduce the effort required to keep the system reliable in different physical infrastructures. Scientific applications can benefit from this approach: For example, grid computing middleware such as Globus Toolkit [5] can be virtualized in VMs of the infrastructure,

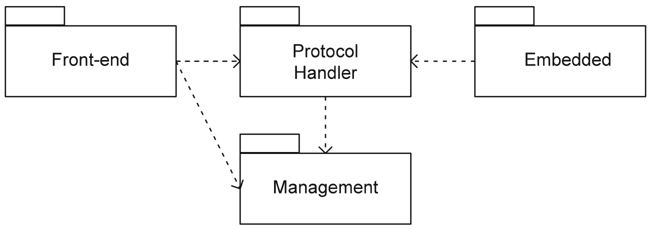


Fig. 13.1 Main packages of the REALcloud platform

reducing the complexity of developing secure intensive computational facilities for massive amounts of data. In robotics, virtualization in cloud is an alternative to keep collaborations between students and to promote robust integration of geographically distant robotic resources.

13.2 Framework for Distributed Cloud Applications

Distributed frameworks must offer sensible SLA and provide high-quality services to concurrent users. In this section, we describe an approach in robotics to develop frameworks associated with scheduling techniques of virtual resources in the design of cloud infrastructures. Extended versions of this work were reported in [6] and [7].

Networked robotics is a trend that favors the distribution of robotic applications across a set of processors located inside and outside robots. The motivation for networked robotics is the availability of network technologies allowing robots to take part in comprehensive networking environments aggregating processors, environmental sensors, mobile and stationary robots, and wireless gadgets, among other networked devices. Many software platforms have been proposed to simplify the development of networked robotic applications, offering a set of services to the applications such as access control, federated authentication, and resource protection. REALcloud is one such cloud platform for networked robotics. Its architecture has four main software packages, as shown in Fig. 13.1.

The embedded package consists of HTTP microservers capable of running on robots' onboard processors with limited processing power. Microservers have an HTTP (Hypertext Transfer Protocol) interface aggregating basic robot operations (move, turn, sense, etc.). The Protocol Handler package intercepts all HTTP requests targeted to the robots and performs functions such as security checks, HTTP proxying, and network address translations. The front-end package offers APIs (Application Programming Interfaces) and Web components for manipulating the robots. APIs are supplied in several programming languages, such as C++, Java, Python, C#, Matlab, and LabView. The management package offers a wide range of services related to users, resources, domains, and federations. An important service is the access service where authenticated users start an access session for the resources they previously reserved.

REALcloud is entirely based on Web technologies. As such, management services and robots are accessed via HTTP. The REALcloud platform has been used primarily in Web labs over the public Internet. In such environments the user develops robotic applications in his/her own computer to control robots over the network. Security is provided by the management and Protocol Handler packages. Although the platform performs adequately for applications requiring small data transferring and processing rates (e.g., sonar-based autonomous navigation), bottlenecks may degrade applications requiring efficient communication and high processing power. Slow Internet connections and HTTP inspections introduce a delay in the control that impairs performance of distributed robotic applications. The processing power of the user's computer also causes delays in control, mainly when control actions are computed via CPU intensive algorithms such as those based on computer vision and computational intelligence techniques.

In order to avoid the delays introduced by slow Internet connections and by limitations of the user's computer, an environment has been developed where user's applications run on servers directly connected to the resources manipulated by the application. The servers can provide resource sharing with much more computer power than the user's processor. Virtualization is the key technology for achieving the desired performance. In addition, applications can take advantage of specialized hardware installed on the servers such as GPUs (Graphics Processing Units) and FPGA (Field-Programmable Gate Array) specialized boards (e.g., for stereo vision processing).

In the case of the networked robotic platform, virtualization helps bringing applications closer to the robots they operate, avoiding long network delays and providing the processing power required by applications. A user can own his/her own VMs with the proper operating system plus the network robotic software necessary for developing and running the applications. This software includes the client side of REALcloud platform, robotic frameworks, APIs, and simulators. Isolation assures that applications running on different VMs do not interfere with each other. This solution requires one or more servers installed in the robotics lab, an inexpensive resource nowadays.

In order to take advantage of virtualization, an architecture must be designed to offer a virtualized environment where the distributed robotic applications will run. In this architecture, resource protection issues must be addressed in order to prevent unauthorized access to robots and other devices by the applications running on VMs. Processor allocation and VM networking sharing are important to assure an adequate distribution of processing power to applications. REALcloud offers the cloud platform as a service in a private (and small) cloud computing infrastructure. Both the client and server sides of the platform are deployed inside VMs. At the server side (management and Protocol Handler packages of Fig. 13.1), virtualization favors software distribution to the members of a federation as all the platform software comes installed and configured in a VM image compatible with a chosen virtualization solution. Each federated domain must deploy instances of this VM to manage and protect the robotic resources.

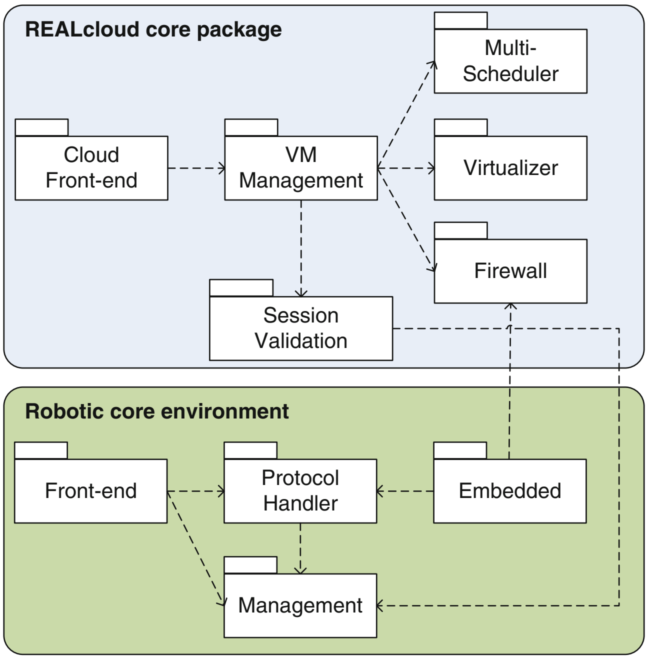


Fig. 13.2 Architecture of the REALcloud framework

At the client side, user's applications running inside VMs access the robotic resources with low communication latency and appropriate computing power. The processors where the VMs run and the robotic resources are connected to the same network or to networks a few hops apart. In order to speed up the interaction with robotic resources, applications running inside VMs access the robotic resources without HTTP inspection by the Protocol Handler package.

The REALcloud environment is built around two Web services (Fig. 13.2): *VM management service* that allows users and administrators to manage VMs and *session validation service* that allows applications running on VMs to access the robotic resources.

The VM management service controls the VM's life cycle. It allows configuring, initiating, reconfiguring, stopping, and destroying VMs. This service relies on command line interfaces supported by the chosen virtualization solution. Once a VM is created, the service configures the VM host's firewall in order to allow access to the VM from outside networks. Access is provided by the NAT (network address translation) and the port forwarding network functions. The session validation service is responsible for assigning privileges to the VMs belonging to users holding valid access sessions. It gives the same protection as provided by the Protocol Handler package (still necessary for accessing resources from the outside networks).

As soon as a user initiates a valid access session, the system creates a session identifier on a Web interface provided by the session validation service. The session validation service queries the cloud access service running in the domain in order to check whether the session ID is a valid one. When the access session terminates, the session validation service reclaims the extra resources allocated to the VM and blocks its access to the resources. Differently from the Protocol Handler package that operates at the application layer, firewalls operate at the network (IP) layer, bringing two important advantages: (1) The decision whether to block or allow the traffic to pass is much faster as it is performed at the packet forwarding level, and (2) any protocol, and not only HTTP/HTTPS, is allowed to pass, as the forwarding decision requires no inspection on the application-level protocol.

13.3 Developing Distributed Applications in the Framework

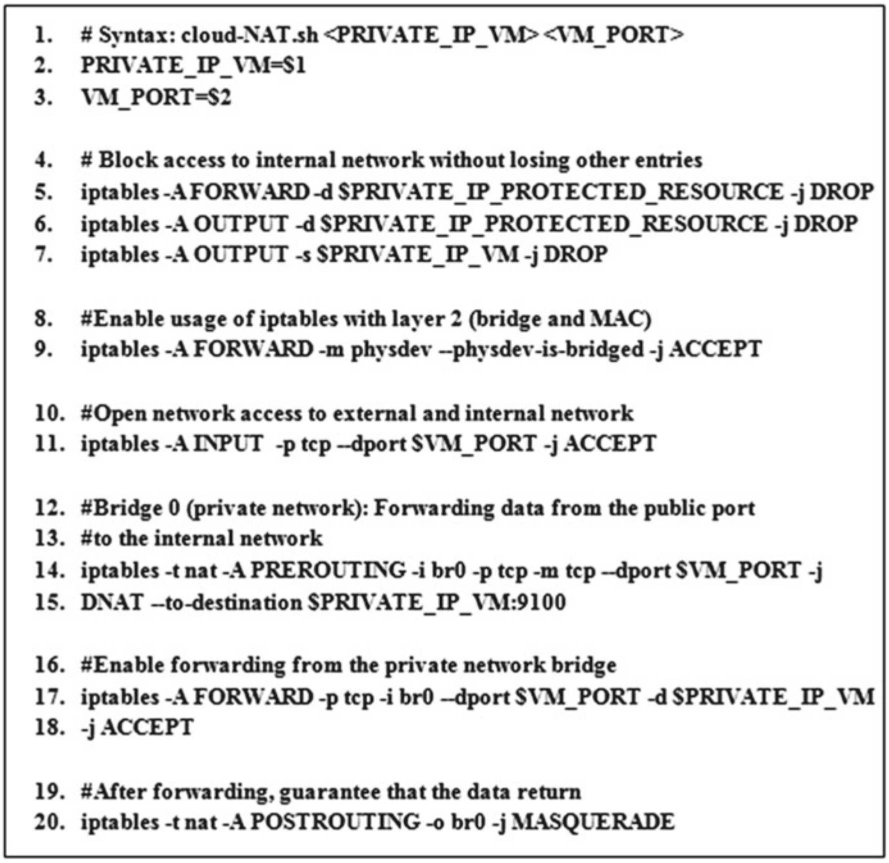
The next steps show how distributed cloud applications can be developed inside the infrastructure according to the features of cloud environments:

Dynamic IP: The VM management component provides dynamic IPs offered by the infrastructure using network bridges. IP table rules are used by the cloud application to establish communication. For instance, the URL "https://staticIP:clientVMPort/," with the same static IP, can be shared by many VMs through network bridges between the server host and the users' VMs. As shown in Fig. 13.3, the following script illustrates how the server host can be configured for this purpose.

Web Services: They are an efficient approach to the development of cloud services. The VM management component can be used to register the Web services provided by the cloud. Services are linked in a REST (Representational State Transfer) approach; that is, each cloud service has a URL accessible by the Internet. Web service methods are available by WSDL interfaces. Remote clients can have access to the Web service functionalities by querying the offered methods in this Web interface. Composition of services can be achieved by the combination of Web services. The communication channel can use SOAP (Simple Object Access Protocol) and can be encrypted by the Axis 2 toolkit [8]. As shown in Fig. 13.4, the following code fragment illustrates how a cloud application can be deployed in the cloud using Axis 2 Web services:

MapReduce Approach: Cloud applications can also be developed according to a MapReduce approach, using pre-configured VMs of the SaaS model. Ready-to-go jobs are another approach to develop distributed cloud applications. Web services can be combined when users' applications are submitted by querying the methods declared in the WSDL interface of the required service.

MapReduce is a programming model geared to the parallel processing of large amounts of data, splitting jobs into a set of independent tasks [9, 10]. It is widely used in searching mechanisms such as Google, Yahoo!, Facebook, Amazon AWS, and Last.fm. The model is noted for its simplicity. A cluster approach is used to distribute and perform the parallel processing of data in multiple cluster nodes,



```
# Syntax: cloud-NAT.sh <PRIVATE_IP_VM> <VM_PORT>
PRIVATE_IP_VM=$1
VM_PORT=$2

# Block access to internal network without losing other entries
iptables -A FORWARD -d $PRIVATE_IP_PROTECTED_RESOURCE -j DROP
iptables -A OUTPUT -d $PRIVATE_IP_PROTECTED_RESOURCE -j DROP
iptables -A OUTPUT -s $PRIVATE_IP_VM -j DROP

#Enable usage of iptables with layer 2 (bridge and MAC)
iptables -A FORWARD -m physdev --physdev-is-bridged -j ACCEPT

#Open network access to external and internal network
iptables -A INPUT  -p tcp --dport $VM_PORT -j ACCEPT

#Bridge 0 (private network): Forwarding data from the public port
#to the internal network
iptables -t nat -A PREROUTING -i br0 -p tcp -m tcp --dport $VM_PORT -j
DNAT --to-destination $PRIVATE_IP_VM:9100

#Enable forwarding from the private network bridge
iptables -A FORWARD -p tcp -i br0 --dport $VM_PORT -d $PRIVATE_IP_VM
-j ACCEPT

#After forwarding, guarantee that the data return
iptables -t nat -A POSTROUTING -o br0 -j MASQUERADE
```

Fig. 13.3 Script for establishment of network bridges

known as worker nodes. The master nodes split the entry data into a set of independent parts (chunks) and distribute them to the worker nodes. A worker node performs a further split, if necessary, in a tree model. Each worker node processes a slice of the main job and forwards its result to the master node. Reduction tasks join the results of one or more worker nodes.

Frameworks to process customized data simplify the development of distributed cloud applications. Hadoop [10] is an example of a framework following the MapReduce model. Hadoop is devoted to homogeneous clusters, and the master node manages the slave nodes with similar configurations. The entry file must be stored in the Hadoop File System (HDFS). This file is split in parts of 64 MB (chunks) by default but can be replicated to reduce fault tolerance. Each chunk is processed by a mapping task that generates a list of<key-value>pairs. The lists are grouped in buckets based on the keys. When each task is processed, reduction tasks are applied to the lists according to the keys. Figure 13.5 is based on [11] and illustrates this model where master and slave nodes can run on cloud VMs.

```
public String getCloudApplicationID(
        String staticIP, String cloudVMPort, String method) {
        String result = "1";
        try {
                CloudStub stub = new CloudStub(
                "https://" + staticIP + “:” + clientVMPort +
                ":/axis2/services/CloudApplicationID");
                CloudStub.GetCloudApplicationID service =
                        new CloudStub.GetCloudApplicationID();
                service.setVirtualMachine(staticIP,cloudVMPort);
                service.setOperation(method);
                CloudStub.GetCloudApplicationIDResponse response =
                        stub.getCloudApplicationID(service);
                result = response.get_return();
        } catch (AxisFault e) {
                result = "Fail in SOAP Axis communication: " +
                e.getMessage();
        } catch (RemoteException e) {
                result = "Remote exception: " + e.getMessage(); }
        return result;
}//end getCloudApplicationID
```

Fig. 13.4 Example of function for Axis 2 Web service

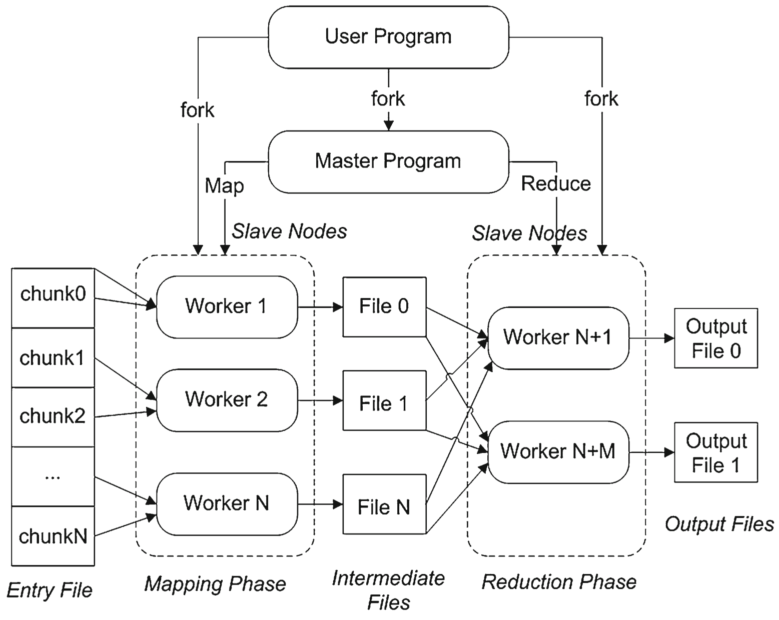
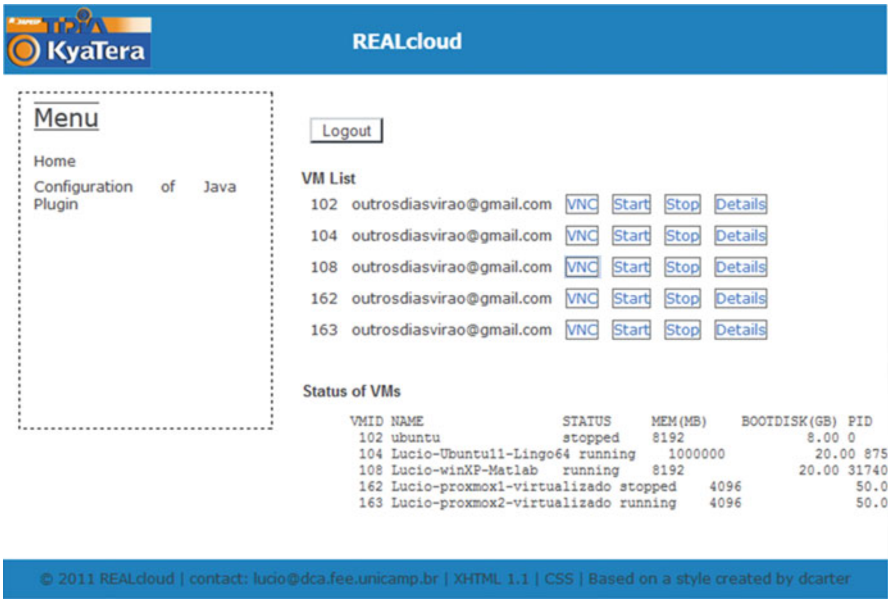


Fig. 13.5 MapReduce for cloud applications in the SaaS model



VM List

102	outrosdiasvirao@gmail.com	VNC	Start	Stop	Details
104	outrosdiasvirao@gmail.com	VNC	Start	Stop	Details
108	outrosdiasvirao@gmail.com	VNC	Start	Stop	Details
162	outrosdiasvirao@gmail.com	VNC	Start	Stop	Details
163	outrosdiasvirao@gmail.com	VNC	Start	Stop	Details

Status of VMs

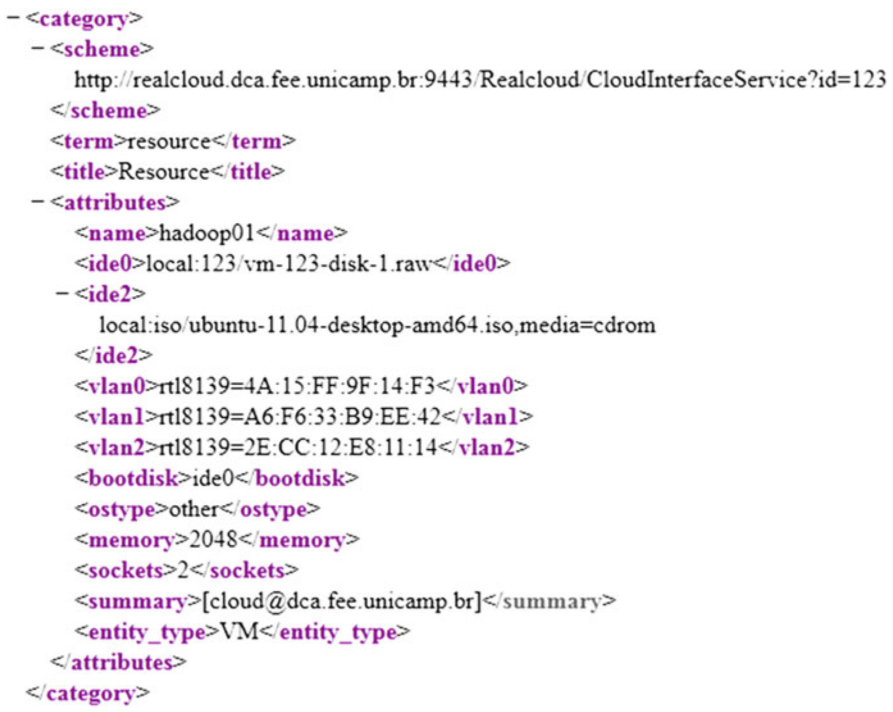
```
VMID NAME                 STATUS     MEM(MB)     BOOTDISK(GB) PID
 102 ubuntu               stopped    8192               8.00 0
 104 Lucio-Ubuntu11-Lingo64 running    1000000            20.00 875
 108 Lucio-winXP-Matlab   running    8192              20.00 31740
 162 Lucio-proxmox1-virtualizado stopped    4096              50.0
 163 Lucio-proxmox2-virtualizado running    4096              50.0
```

© 2011 REALcloud | contact: lucio@dca.fee.unicamp.br | XHTML 1.1 | CSS | Based on a style created by dcarter

Fig. 13.6 Web client application based on OCCI specifications

Open Cloud Computing Interface (*OCCI*)*-Based Infrastructure*: OCCI is a set of specifications maintained by the Open Grid Forum (OGF) to define interfaces to deliver cloud resources. OCCI is a RESTful protocol and API for management tasks acting as a service front-end to a provider's internal management framework. The standards are described in three documents: OCCI Core [12] describes the formal definition of the OCCI Core Model; extensions in this API will be discoverable and visible to an OCCI client at run-time. OCCI Infrastructure [13] defines the model to extend the IaaS and describes resource types, their attributes, and actions over them. OCCI HTTP Rendering [14] defines the mechanism to access the OCCI Core Model in a RESTful approach using the HTTP protocol.

As an example, the REALcloud infrastructure offers a set of Web services for the development of new cloud applications and HTTP syntax for the dynamic discovery of the available users' virtual machines. "https://cloudStaticIP:cloudPort/Realcloud/resources.jsp?action=<VNC|START|STOP|DETAILS>&resourceName=VM_ID" is the URL to interact with the set of actions of the user virtual machine. HTTP queries are used to start a VNC session between the client Web browser and the cloud environment. The other actions are to start, stop, and query details about each virtual machine of the authenticated user. Figure 13.6 shows the Web client interface; a RESTful approach with HTTP queries is also available. This option is important to acquire management information about all virtual machines in the cloud environment. As shown in Fig. 13.7, the URL "https://cloudStaticIP:cloudPort/CloudInterface?id=VM_ID" returns the OCCI-based XML data.



```
- <category>
  - <scheme>
      http://realcloud.dca.fee.unicamp.br:9443/Realcloud/CloudInterfaceService?id=123
    </scheme>
    <term>resource</term>
    <title>Resource</title>
  - <attributes>
      <name>hadoop01</name>
      <ide0>local:123/vm-123-disk-1.raw</ide0>
    - <ide2>
        local:iso/ubuntu-11.04-desktop-amd64.iso,media=cdrom
      </ide2>
      <vlan0>rtl8139=4A:15:FF:9F:14:F3</vlan0>
      <vlan1>rtl8139=A6:F6:33:B9:EE:42</vlan1>
      <vlan2>rtl8139=2E:CC:12:E8:11:14</vlan2>
      <bootdisk>ide0</bootdisk>
      <ostype>other</ostype>
      <memory>2048</memory>
      <sockets>2</sockets>
      <summary>[cloud@dca.fee.unicamp.br]</summary>
      <entity_type>VM</entity_type>
    </attributes>
  </category>
```

Fig. 13.7 OCCI-based document obtained in the RESTful HTTP query

Multi-scheduler Infrastructure: The multi-scheduling approach employs different scheduling algorithms to distribute cloud resources according to resource features such as CPU availability, RAM usage, and storage capacity. Many cloud solutions use multi-scheduling approaches to optimize usage of their shared resources [15].

Eucalyptus [16] employs an allocation resource process dispatched by the cloud provider, which ends when the requested VM is instantiated in a network node. When an allocation request is placed, the CLC (cloud controller) component determines which CC (Cluster Controller) component will be able to instantiate the VM. This is done by querying for cloud resources and selecting the first CC component that has available resources.

Nimbus [17] manages its resources by means of the Workspace Resource Manager component. It gives the cloud developer control over manageable node groups using the libvirt library [18], jointly with the Workspace Pilot component, which receives user jobs and performs scheduling with additional schedulers, such as Condor [19].

REALcloud uses a multi-scheduling approach similar to OpenNebula [20], as shown in Fig. 13.8: an embedded default scheduler with a rank algorithm to distribute its VMs according to VM requirements and the servers' performance. The pseudo-code below shows the algorithm for resource allocation. The parameters used for entry requests are host, CPU and RAM availability, and type of hypervisor.

```
Algorithm for Resource Allocation in REALcloud
Required: Input: requirements, rank, hostList
Ensured: Output: selectedHost
  while (hostList.hasElements) do
    if host.satisfies(requirements) then
      candidates.new(host)
    endif
  endwhile
  sort = sort(candidates,rank)
  selectedHost = sort(1)
```

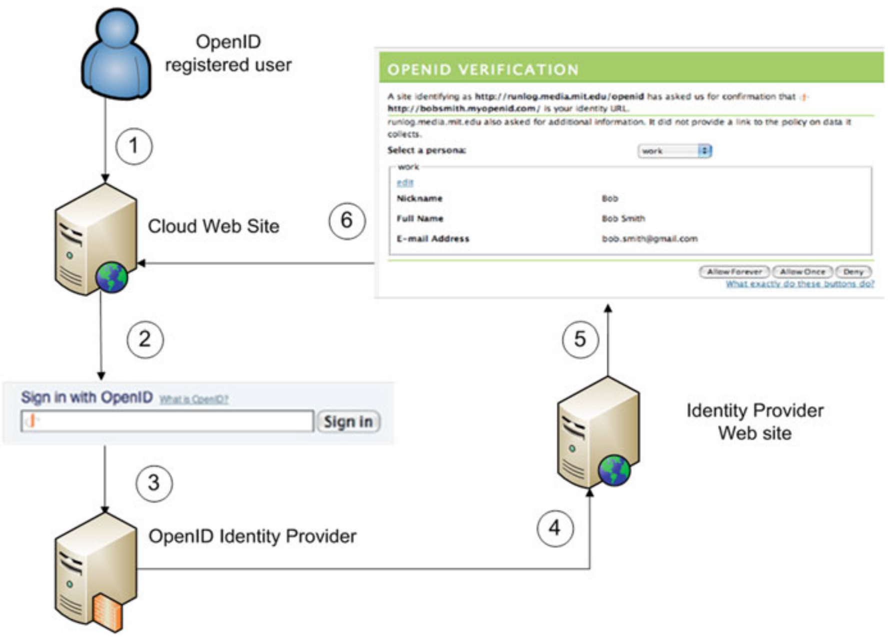
Fig. 13.8 Algorithm for resource allocation in REALcloud

The rank function sorts hosts according to their availability and the users' requirements to instantiate VMs. New scheduling algorithms can be implemented based on this policy.

Identity Management with OpenID: Public cloud services of authentication and authorization can be aggregated into private clouds in a model known as hybrid cloud, a combination of public and private cloud models.

This approach is useful to avoid keeping large databases in the internal infrastructure; that is, valid users in trusted domains can be authenticated in the private cloud. However, authorization must be managed by the internal private infrastructure. This approach can be used in the cloud front-end package. Figure 13.9 shows the basic authentication mechanism with OpenID. OpenID is a passive protocol that uses HTTP forwarding between users' applications and the identity provider. Requests to access the authentication service are based on HTTP protocol. Users must first register themselves in an identity provider with OpenID support, which in turn uses the user account to generate a unique URL in the Web. The URL is used by the client's application as an argument to discover the authentication service; that is, authentication is a service provided by the identity provider. This URL is used by the client application to query the identity provider that keeps the user's account. In the following step, users not previously authenticated must provide their credentials (typically, user ID and password) to the authentication service of the identity provider, identified by URL. OpenID also has mechanisms to delegate rules between peers of the same circle of trust.

At step 1, a user with a registered identity in an OpenID provider (Google account, for instance), but not previously registered in this domain, wants to access resources in a cloud Web site having an OpenID authentication service. In step 2, the user enters the OpenID URL that he/she received from the identity provider. In step 3, the OpenID service of the cloud Web site redirects the user's browser to the authentication service of its identity provider. In step 4, two options are available: If the user has been previously authenticated in the identity provider, the browser is redirected to the validation service of the cloud Web site. If the user has not been previously authenticated, the identity provider queries about credentials (user ID and password) to proceed with browser redirecting. In step 5, the identity provider



A site identifying as **http://runlog.media.mit.edu/openid** has asked us for confirmation that **http://bobsmith.myopenid.com/** is your identity URL.

runlog.media.mit.edu also asked for additional information. It did not provide a link to the policy on data it collects.

Select a persona: work

work

edit

Nickname	Bob
Full Name	Bob Smith
E-mail Address	bob.smith@gmail.com

Fig. 13.9 Basic authentication with OpenID

Web site uses a verification service to validate the URL address that queries about authentication with OpenID. This step is necessary for security reasons to avoid phishing attacks (untrusted URL address). Another reason is that many sites want to have additional information about newer users, such as user name, alternative e-mail, and telephone number. In step 6, the identity provider redirects the user to the cloud resource URL.

13.4 Overview of Cloud Distributed Environments

The complete hybrid cloud environment was developed to support many concurrent users by simplifying the usage of virtual machines inside and outside the infrastructure while keeping the requirements of availability, reliability, network performance, and security of the whole system. This section describes the architecture, APIs, and methodology to develop distributed applications in this environment.

Figure 13.10 shows the main components of a generic cloud computing environment. According to this model, more specific environments can be implanted by specializing each component. There is no clear rule requiring the use of all components, but more complete environments should recognize their main parts in this model. A description of each one follows.

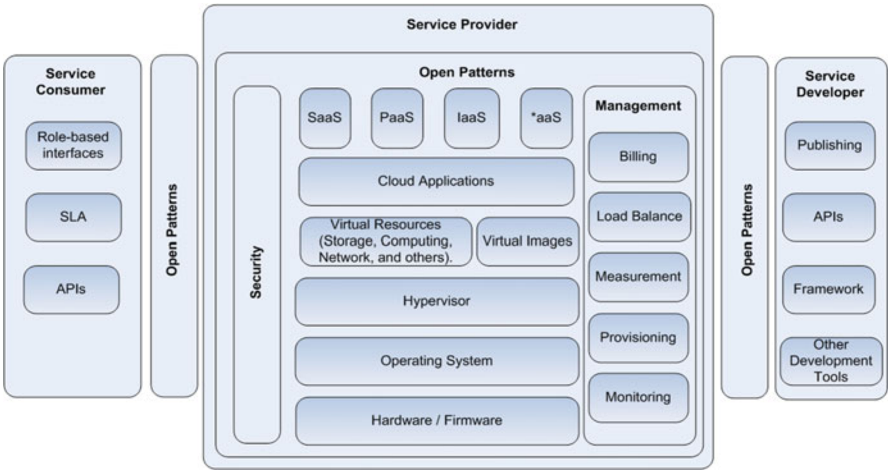


Fig. 13.10 Overview of the main components of cloud computing environments (Based on [21])

13.4.1 Service Provider

This component contains the main elements that make this environment functional. The versatility of the *service provider* component is supported by many open source solutions, mainly to increase the possibility of linkage with other cloud interfaces, extending and developing other compliant components without commercial restrictions, and reducing the usage of closed patterns. The bottom level of this core model defines the *hardware/firmware* component. Distinct environments are highly dependent on the base infrastructure. Server architecture (e.g., x86 or x64) and the availability of virtualization in hardware (e.g., CPUs with registers to support virtualization) can have a direct impact on the performance of the whole cloud system. The hardware includes physical servers, routers, switches, storage devices, backup drivers, and firewalls. Each communication device in this base network is offered by the datacenter provider.

Selecting the *operating system* is important because the type of virtualization will depend on it. Proprietary operating systems are regulated by commercial licenses, increasing implantation costs in future security updates and/or software expansion. Open source solutions bring the advantage of cost reduction, but the type of management service must use APIs compatible with the type of virtualization solution. Finally, the provider must consider the need of dedicated servers. For example, cloud solutions such as Xen XCP [22] use dedicated hosts to offer their services.

The *hypervisor* is the software layer between the hardware and the operating system and is responsible for offering shared resources to large numbers of concurrent virtual machines (VM). The hypervisor runs in supervisor mode and manages the scheduling of resources (CPU cycles, memory slices, disk storage, network linkage, and so on) offered by the base infrastructure. Hypervisors intercept requests from VMs and emulate privileged instructions. Hypervisors running directly over the hardware are known as type 1 (e.g., Xen), and those running above the operating system are known as type 2 (e.g., VirtualBox [23]).

The hypervisor is also responsible for emulating *virtual resources* such as I/O devices, CD/DVD drives, mouse, keyboard, and network interfaces. This component must look after several security issues. Vulnerabilities in memory access security rules in the hypervisor can lead to unauthorized access to the virtual machine, compromising data reliability. Cloud solutions such as Abiquo [24] and OpenNebula support many hypervisors, each kept in a different server host. In this case a complementary management of these cloud nodes is necessary. These virtual resources are provided by the management component. The management component also offers other resources such as storage, complementary features to computing (e.g., more cycles/cores of CPU, RAM), network bandwidth, and Network File System (NFS).

Much of the success of cloud computing is related to the rapid development of virtualization techniques, accomplished by technical advancements and cost reduction in computational hardware. *Virtual machines* (VMs) are an example of the success of this theme. Many VMs can be instantiated in the same server host, helping reduce the number of physical servers by means of a more efficient usage of resources, a technique known as server consolidation. A complete operating system can be installed inside a VM, which in turn can be distributed or migrate to another server host. Migration is possible if the destination server host has a compatible virtualization interface. The format of different VMs can be converted to other formats if the virtualization toolkit provides this feature, contributing to distribute "ready-to-use" systems to distinct cloud providers. In addition, many cloud providers, such as Amazon EC3 and GoGrid [25], and cloud solutions, such as Eucalyptus, OpenNebula, and Abiquo, provide templates of pre-configured VMs for their environments.

Cloud applications are inherently distributed applications with interfaces to interact with the services provided by the cloud. The main consideration in their design is that these applications have to be supplied by the cloud environment, whether using virtualized services or any other technology with Internet access such as HTTP or SOAP. Distributed cloud applications are different from conventional applications with remote access, mainly because the environment has the features of [26]:

Elasticity: Shared resources should be provided to cloud applications on demand, that is, as soon as the cloud applications need them, but only for the period of usage. The cloud management system should reallocate non-used resources when the applications no longer need them.

High access level to computational resources: Cloud applications should be accessible by a gamut of different remote devices: laptops and desktops, mobile phones, smart phones, tablets, and so on.

Multi-tenant behavior: The same cloud application can be used by multiple users (tenants). This model is valuable because multiple client applications can share the same remote application. A single instance of the software runs on the server, providing services to many concurrent client applications.

Transparency: Cloud applications are offered independently from their physical location, and although users need not care about where their applications run inside the cloud, this information should be given by the cloud provider. Legal restrictions in some countries do not allow some particular contents to be provided in their geographical location and/or jurisdiction.

Pay-per-use model: Billing is proportional to the usage of computational resources, similarly to traditional bills of electricity, water, and natural gas.

Scalability: Consumption of shared computational resources or the increase of cloud applications and users should not degrade the performance of other concurrent cloud applications in the same domain. This issue is a consequence of the elasticity model.

Different models to provide service are described in the literature:

PaaS (*Platform as a Service*): Users can develop their own applications with toolkits provided by the cloud platform. Communication services are also available, for example, Web services, storage, and programming languages. Examples are Ning [27] and Microsoft Windows Azure Platform [4].

SaaS (*Software as a Service*): The cloud provider enables usage of exclusive user applications and/or applications provided by the cloud environment, such as enterprise e-mails, discussion groups, Web site toolkits, and workflow applications. Examples are Salesforce [28] and Google Apps [29].

IaaS (*Infrastructure as a Service*): Computational resources such as storage, high-performance computing (HPC), high network bandwidth, logical servers, and a set of other resources and devices are provided by the infrastructure. Examples are Amazon AWS and FlexiScale [30].

**aaS* (*Everything as a Service*): Any services and/or application available in a cloud model such as a combination of the previously cited models.

13.4.2 Security

The main issues about security can be grouped according to their importance to the software-level (cloud applications) and to the hardware-level infrastructure. These issues should be addressed by each element in the service provider component.

Software-level security deals with the role of the communication protocol in the privacy, integrity, and authentication in interactions with cloud applications [31].

Privacy exists when only sender and receiver are able to understand the communication. If someone eavesdrops on the communication channel, its contents should not be understood by the third party.

Integrity is guaranteed when the receiver can be sure that he/she acquired the message exactly as sent by the other party.

Authentication is relevant because it increases the security access level to cloud services. Over the Internet, the HTTPS protocol, session cookies, and X.509 certificates are options to guarantee the end-to-end privacy between cloud services and their users.

Communication with SSL uses a secure channel to forward data between the server and the client application. An authenticated channel can be built using digital signatures and a public key infrastructure. In addition, the cloud management system should be able to provide tools to manage the authentication of its users to ensure confidentiality, as well as authorization techniques (e.g., role-based access control – RBAC) to differentiate the access to services [32, 33]. If the software is provided by or developed in the cloud, the platform needs to keep policies to ensure that no harmful software, such as worms, trojans, or viruses, can propagate in the system.

Security for infrastructure deals with the guarantee that access to cloud resources is protected against external malicious users. Generally this can be achieved by firewall rules between the public link access and the private cloud network (e.g., using IP table rules). Resource availability should be managed with techniques of fail tolerance, load balance, patch management, monitoring, backup redundancy, and others. However, this whole set of techniques will only be effective if clear rules are kept to control personal access to physical hosts.

13.4.3 Service Consumer

Cloud users have access to cloud services by interfaces compatible with the cloud environment. *Role-based interfaces* allow different interactions with the cloud services according to the role that each specific user plays in the environment. For example, authenticated users must be able to log into the system, instantiate/stop VMs, perform status queries, and so on, but administrative functions such as creating and removing VMs should be restricted to them. This same issue is seen in collaborative applications such as Google Docs [34] and Picasa [35], where the RBAC roles are applied to users.

Service-Level Agreements (*SLAs*) should be regulated by the law of the country. In scenarios where agility to accommodate unpredictable consumption is important, SLAs are critical to define the relationship between the cloud service provider and its consumers. A more detailed report of this issue can be found in [36].

Application Programming Interfaces (*APIs*) on the side of the service consumer must also abide by the rules when interacting with remote services. APIs simplify the development of new services, but the cloud provider must keep its APIs up to date to avoid security risks.

13.4.4 Management

Billing follows the pay-per-use model, in which the price charged is proportional to resource consumption. OpenQRM [37] is an open source example of architecture that allows billing in the private cloud and supports EC2 standards for APIs. It also supports virtualization techniques such as KVM [38] and Xen, as well as management of hosts, virtual machines, and deployments. Virtualized images of Ubuntu, Debian, and CentOS are supplied as templates. However, in many other private cloud solutions, the billing component is not necessary.

Load balance deals with how the infrastructure supports requests and how its resources are maintained to achieve high performance and better utilization.

The *measurement* component establishes metrics to perform several management tasks.

Provisioning deals with policies to offer resources to many concurrent users. Again, policies must take into account availability, scalability when more resources need to be provided by other domains, and resource scheduling. It is common for each cloud solution to implement its own solutions to monitoring, but this task can be carried out with open source middlewares, such as Nagios [39], an open source tool allowing extensions by plug-ins. For instance, the NRPE (Nagios Remote Plugin Executor) monitors the number of users logged in the system, CPU consumption, memory used by each virtual machine, and number of active processes in the server hosts [40].

13.4.5 Service Developer

Publishing describes how services are provided and how they can be accessed, either internally or over the Internet. For example, access to virtual machines can be provided by a specific URL and/or via VNC protocol. In addition, applications can show their methods in WSDL language, and communication can be done via the HTTP or SOAP protocols. Many providers offer their own sets of *APIs* (e.g., Google App Engine) to interact with their public services according to the PaaS model. Also provided are exclusive *frameworks*, for example, Microsoft Azure with .NET framework, and *other development tools*, for example, datasheets, corporate e-mail, workflows, and other tools in Salesforce.com.

13.5 Final Considerations

It is important that the development of cloud applications be guided by frameworks to avoid a mix of unrelated structures. The main features of cloud domains need to be considered jointly with the needs of the institution. Furthermore, open standards contribute to simplifying the integration with other domains and extending the portability of applications.

Related issues in the development of cloud applications are about collaborative applications such as Google Docs, storage in cloud with Dropbox, and Google Drive. Such applications are highly dependent on network performance between the client user and the service provider. In addition, cache routines in the client application guarantee data integrity.

Many other security features aim to increase the reliability of data exchange. Synchronization protocols are an example – the timestamp needs to be valid in both sides. Network data encryption with AES 256 bit and SSL connection are extra protection offered by some providers.

Portability is another issue to be considered. Customizing the cloud service according to the client device features is another challenge, for example, for Web connection with mobile devices.

Much research has been done on how to provide inter-cloud communication and establish federations [41]. Cloud computing is emerging as a new paradigm to offer services in the Web, one that can lead to new business opportunities, but the difficult issue of security remains open. This is because in a cloud numerous applications are available as services, many of which have their own access control systems. Furthermore, applications that support service compositions across distinct domains require authentication mechanisms that take into account this collaborative nature.

Acknowledgments The author thanks Marcos Favero Florence de Barros for his language editing services.

References

1. Open Cloud Computing Interface (OCCI): Available at: http://occi-wg.org/about (2012)
2. OpenID Foundation Website: Available at: http://opened.net (2012)
3. Amazon AWS. Amazon Elastic Compute Cloud (Amazon EC2): Available at: http://aws.amazon.com/ec2 (2012)
4. Windows Azure: Microsoft's Cloud Platform: Available at: http://www.microsoft.com/windowsazure (2012)
5. Globus Toolkit: Welcome to the Globus Toolkit Homepage. Available at: http://www.globus.org/toolkit (2012)
6. Rocha, L.A., Olivi, L., Feliciano, G., Paolieri, F., Rodrigues, D., Cardozo, E., Guimarães, E.: A cloud computing environment for supporting networked robotics applications, DASC. In: IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp. 1110–1116. Sydney, Australia (2011)
7. Rocha, L.A., Feliciano, G., Olivi, L., Cardozo, E., Guimarães, E.: A bio-inspired approach to provisioning of virtual resources in federated clouds, DASC. In: IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp. 598–604, Sydney, Australia (2011)
8. Apache Axis2: Available at: http://axis.apache.org/axis2 (2012)
9. Shankar, R., Narendra, G.: MapReduce programming with Apache Hadoop – process massive data sets in parallel on large clusters. Javaworld. Available online at: http://www.javaworld.com/javaworld/jw-09-2008/jw-09-hadoop.html (2008)
10. Yahoo! Developer Network: MapReduce: Available at: http://developer.yahoo.com/hadoop/tutorial/ (2012)
11. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
12. GFD.183 – OCCI Core (v1.1): Available at: http://www.ogf.org/documents/GFD-183.pdf (2011)
13. GFD.184 – OCCI Infrastructure (v1.1): Available at: http://www.ogf.org/documents/GFD-184.pdf (2011)

14. GFD.185 – OCCI HTTP Rendering (v1.1): Available at: http://ogf.org/documents/GFD-185.pdf (2011)
15. Gonçalves, G.E., Endo, P.T., Cordeiro, T.D., Palhares, A.V.A., Sadok, D., Kelner, J., Melander, B., Mangs, J.: Resource allocation in clouds: concepts, tools and research challenges. In: Minicurso– SBRC. Campo Grande, MS, Brazil (2011)
16. Johnson, D., Murari, K., Raju, M., Suseendran, R.B., Girikumar, Y.: Eucalyptus Beginner's Guide – UEC Edition – Ubuntu Server 10.04 – Lucid Lynx. CSS Corp. Available online at: http://cssoss.files.wordpress.com/2010/06/book_eucalyptus_beginners_guide_uec_edition1.pdf (2010)
17. Nimbus Project: Available at: http://www.nimbusproject.org/ (2012)
18. Libvirt – virtualization API: Available at: http://www.libvirt.org (2012)
19. Condor High Throughput Computing: Available at: http://research.cs.wisc.edu/condor/ (2012)
20. OpenNebula Project Leads: Opennebula. Available at: http://opennebula.org(2012)
21. Amrhein, D., et al.: Cloud Computing Use Cases White Paper Version 4.0. Technical Report (2010)
22. XenServer: Available at: http://www.citrix.com (2012)
23. VirtualBox: Available at: http://www.virtualbox.org (2012)
24. Abiquo: Architecture Overview: Available at: http://community.abiquo.com (2012)
25. GoGrid: Available at: http://www.gogrid.com (2012)
26. Martins, A.: Fundamentos de Computação Nuvem para Governos – Amãpytuna – Computaç ão em Nuvem: serviços livres para a sociedade do conhecimento, chapter 2, pp. 47–65. ISBN: 978-85-7631-241-3. Alexandre de Gusmão Foundation (2010)
27. Ning: Available at: http://www.ning.com (2010)
28. Salesforce: Available at: http://salesforce.com (2012)
29. Google Apps for Business: Available at: http://www.google.com/a/ (2012)
30. FlexiScale public cloud: Available at: http://www.flexiant.com/products/flexiscale (2012)
31. The Globus Toolkit 4 Programmer's Tutorial: Fundamental Security Concepts. The three pillars of the secure communication. Available at: http://gdp.globus.org/gt4-tutorial/singlehtml/progtutorial_0.2.1.html (2005)
32. Ramachandran, M.: Component-Based Development for Cloud Computing Architectures. Cloud Computing for Enterprises Architectures, Computer Communications and Networks. Springer, London (2011)
33. Ahmed, K.E.U., Alexandrov, V.: Identity and Access Management in Cloud Computing. Cloud Computing for Enterprises Architectures, Computer Communications and Networks. Springer, London (2011)
34. Google docs: Available at: http://docs.google.com (2012)
35. Picasa Web Albums: free photo sharing from Google: Available at: http://picasaweb.google.com (2012)
36. Buck, K., Hanf, D.: Cloud SLA Considerations for the Government Consumer. Systems Engineering at MITRE. Cloud Computing Series. The MITRE Corporation. Available online at: http://www.mitre.org/work/tech_papers/2010/10_2902/cloud_sla_considerations_government.pdf (2012)
37. OpenQRM: Available at: http://www.openqrm-enterprise.com/ (2012)
38. Kernel-based Virtual Machine: Available at: http://www.linuxkvm.org/page/Main_Page (2012)
39. Nagios – The Industry Standard in IT Infrastructure Monitoring: Available at: http://www.nagios.org (2012)
40. Chaves, S.A., Uriarte, R.B., Westphall, C.B.: Implantando e Monitorando uma Nuvem Privada. In: VIII WCGA, Brazilian Symposium on Computer Networks and Distributed Systems, SBRC. Gramado, RS, Brazil (2010)
41. Buyya, R., Ranjan, R., Calheiros, R.N.: Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: challenges and opportunities, high performance computing & simulation. In: HPCS '09. International Conference, Leipzig (2009)