J Internet Serv Appl (2011) 2:243–255
DOI 10.1007/s13174-011-0035-x

SI: CLOUD COMPUTING

# Toward role-based provisioning and access control for infrastructure as a service (IaaS)

**Dongwan Shin · Hakan Akkan · William Claycomb · Kwanjoong Kim**

Received: 9 November 2010 / Accepted: 10 August 2011 / Published online: 8 September 2011


**Abstract** Cloud computing has drawn much attention in recent years. One of its service models, called infrastructure as a service (IaaS), provides users with infrastructure services such as computation and data storage, heavily dependent upon virtualization techniques. Most of the current IaaS providers take the *user-resource direct mapping* approach for their business, where individual users are the only type of service consumer who can request and use virtualized resources as long as they pay for the usage. Therefore, in this approach, the users and virtual resources are centrally managed at the IaaS providers. However, this also results in the lack of support for scalable authorization management of users and resources, organization-level policy support, and flexible pricing for business users. Considering the increasing popularity and growing user base of cloud computing, there is a strong need for a more flexible IaaS model with a finer grained access control mechanism than the aforementioned *all-or-nothing* approach. In this paper we propose a domain-based, decentralized framework for provisioning and managing users and virtualized resources in IaaS. Specifically, an additional layer called *domain* is introduced to the user-resource direct mapping scheme, whereby de-centralization of user and resource management is facilitated. Our framework also allows the IaaS service provider to delegate its administrative routines to domains so that each domain is able to manage its users and virtualized resources allocated by the IaaS provider. Our domain-based approach offers benefits such as scalable user/resource management, domain-based security and governance policy support, and flexible pricing.

**Keywords** Cloud computing · IaaS · Domain-based · Decentralized cloud · Role-based access control

This article is an expanded version of the paper "Domain-based Virtualized Resource Management in Cloud Computing" which appeared in the Proceedings of 5th International Workshop on Trusted Collaboration (TrustCol 2010).

D. Shin (✉) · H. Akkan
Secure Computing Laboratory, New Mexico Tech, Socorro, NM, USA
e-mail: doshin@nmt.edu

H. Akkan
e-mail: hakkan@nmt.edu

W. Claycomb
Sandia National Laboratories, Albuquerque, NM, USA
e-mail: wclayc@sandia.gov

K. Kim
Hanseo University, 360 Daegok-li, Seosan-si, Korea
e-mail: kimkj@hanseo.ac.kr

## 1 Introduction

Cloud computing is a new type of computing, which enables convenient, on-demand access to computing resources. Though its definitions, attributes, and characteristics are varied, it has been rapidly emerging and diversified, and is being adopted as a new potential infrastructure for enterprise, government, and academic computing. Generally, it is considered to be a model that promotes the availability of computing resources, which can be rapidly provisioned and serviced with minimal management effort or service provider interaction [16]. Typical examples of cloud computing include office applications migrated to the Internet such as Google Docs, and enterprise computing & storage service such as Amazon EC2 & S3, Google App Engine, Salesforce's Cloud Platform, and Microsoft's Azure [2, 10, 22, 29].

With major industry players like Amazon, Google, IBM, and Microsoft, the federal and state governments have also shown keen interest in cloud computing; for instance, the U.S. Census Bureau is using Salesforce's cloud services to manage the activities of about one hundred thousand partner organizations across the country; the Defense Information Systems Agency (DISA) has a private cloud within its data centers which provides human resource management services to both U.S. Army and Air Force; and NASA Ames Research Center recently announced the development and deployment of a cloud computing infrastructure called Nebula [15], to provide high-capacity computing and storage services by using a virtualized and scalable approach to achieve cost and energy efficiency.

One of the service models of cloud computing [16], called infrastructure as a service (IaaS), provides users with infrastructure services such as computation and data storage, and virtualization is one of the fundamental techniques enabling this model. Specifically, infrastructure resources are provisioned on virtual platforms and provided as an on-demand service to users, and this offers benefits such as elasticity, and cost/energy efficiency. Most of the current IaaS providers take a *user-resource direct mapping* approach for their business, where the individual user is the only type of service consumer who can request and use virtualized resources with a *root* privilege as long as they pay for the usage. Hence, in this approach, management of users and virtualized resources is centralized and administered with relative ease at the IaaS providers. However, this also results in the lack of support for scalable authorization management of users and resources, organization-level policy specification and enforcement, and flexible pricing for business users. Considering the increasing popularity and growing user base of cloud computing, there is a strong need for a more flexible IaaS model with a finer grained access control mechanism than the aforementioned *all-or-nothing* approach.

### 1.1 Motivation example

IaaS platforms such as Amazon EC2 provide computing resources as a metered service similar to a traditional public utility such as electricity and water. Hence, it is also called as utility computing. Let us walk through a simple example aimed at illustrating this utility computing aspect of IaaS, and we will use it throughout this paper.

*Example 1* Assume that **SunnyTech.edu**, a fictitious university, decides to transition its university-wide information technology (IT) as well as department-wide research, teaching, and service IT to the cloud services provided by **GoCloud.com**. The cloud services under consideration include not only software services such as email and human resource (HR) applications, but also infrastructure services such as virtual machines with operating systems (OSes) having preconfigured application images installed for various teaching and research purposes. In addition, **SunnyTech.edu** wants to control access to, and thus limit the usage of, the virtualized resources based on the roles of the members of the university. Lastly, the university wants to not only keep access logs based on its audit policy but also implement other security and governance policies.

Assuming that **GoCloud.com** is based on the user-resource direct mapping approach like Amazon EC2 cloud service, several critical questions can be raised. First, how can **SunnyTech.edu** support role-based access control (RBAC) [24] for controlling access to virtualized resources for its faculty, students, and employees? The egalitarian approach taken by **GoCloud.com** would not easily allow the university to support that feature without major system modifications. Second, how about the university's ability to specify its own security and governance policies? Since there is no concept of organization in the user-resource direct mapping approach, it would be really difficult to support organization-based policies without **SunnyTech.edu**'s costly effort to multiplex all requests from its members through a single channel to the cloud service provider. Lastly, there is no category for business customers in the current pricing model of **GoCloud.com**. As a result, it seems hard for **SunnyTech.edu** to expect a special discount rate, which is a very common pricing practice for most of the traditional utility companies.

We believe that introducing an intermediary into the direct user-resource mapping approach could help us address the issues raised above. The intermediary can be a collection of users, a collection of virtualized resources, or a collection of both users and virtualized resources. The *reserved instances* in Amazon EC2 can be considered to be the second type of intermediary: a collection of virtualized resources. Though the remaining two can both be considered to be another type of service consumer for the IaaS provider, what we are interested in this paper is the third type of intermediary: *a collection of both users and virtualized resources*. Hence, managing users and resources within the intermediary can be done independently of the IaaS provider, and an advanced access control mechanism along with other security and governance policies can be implemented and applied within the intermediary.

### 1.2 Objective and organization

In this paper we introduce the notion of domain as an intermediary into the direct mapping approach taken by most of the current IaaS providers. The domain pertains to any organization using the cloud services as a business user. Our domain-based framework facilitates decentralized management of users and virtualized resources in IaaS. Specifically,

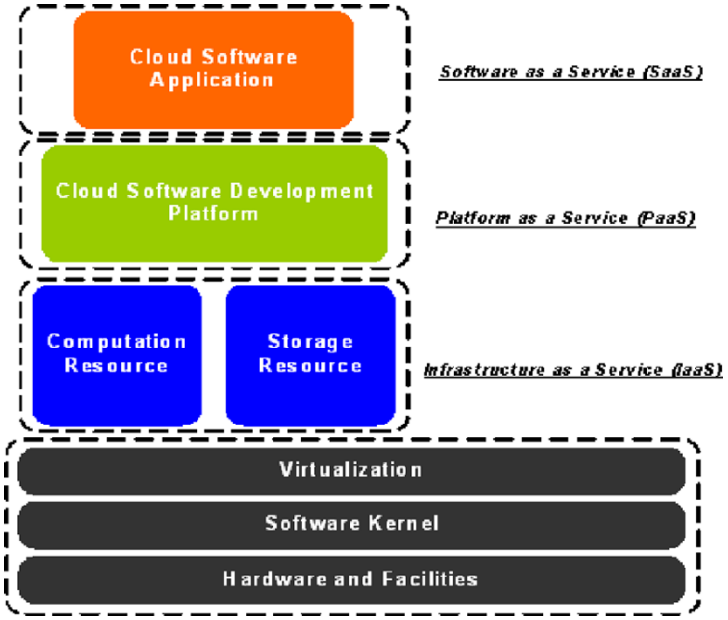


**Fig. 1** Cloud computing components and delivery models

our approach enables the cloud service provider to delegate its administrative works involving users and resources to domains so that each domain is able to manage its own users and virtualized resources allocated from the cloud service provider. In addition, each domain is allowed to specify and enforce its own security and governance policies. Therefore, our framework provides such benefits as scalable user authorization management, domain-based security and governance policy support, and flexible pricing for business users.

Our contributions in this paper are as follows: first, the concept of using domains is new for most of the existing IaaS providers and there seems to be no previous approach that is similar to ours in literature yet. Second, to leverage the benefits offered by the introduction of domains, we present how to inject a role-based access control and provisioning scheme to IaaS. Lastly, we provide a prototype implementation based on and extended from an existing open source IaaS platform called Eucalyptus [7] to test the feasibility of our approach.

The rest of this paper is organized as follows. Section 2 discusses background and related work. Section 3 describes our approach to domain-based user and resource management, followed by the discussion of our design along with a security architecture in Sect. 4. Section 5 discusses our implementation with performance analysis. Section 6 concludes this paper with a discussion on our future research direction.

## 2 Background and related work

In this section we first discuss the general characteristics of cloud computing along with the discussion of its three different service models. Then, we investigate the current support of role-based access control as a way to ease the administration and management of user privileges in cloud-based platforms as well as distributed computing platforms.

### 2.1 Cloud computing and its service models

The general characteristics of cloud computing include on-demand service, ubiquitous access, location independence, rapid elasticity, and measured service [16]. To support these characteristics, cloud computing generally consists of three foundational components and three optional, applied components depending on its service models. The three foundational components are essentially those that are needed to build a collection of physical/virtualized, distributed servers for providing cloud services. They are (1) hardware and facilities, (2) software kernel, and (3) virtualization, as shown in the lower part of Fig. 1. The three applied components characterize and classify the services and applications of cloud computing. They are (1) computation and storage resource, (2) cloud software development platform, and (3) cloud software application, as shown in the upper part of Fig. 1. The computation and storage resource component is concerned with the service model called *Infrastructure as a Service (IaaS)*, the cloud software development platform component pertains to another service model called *Platform as a Service (PaaS)*, and the cloud software application component is related to the service model called *Software as a Service (SaaS)*. Their differences are as follows; first, in the SaaS model, the cloud consumer can use the cloud provider's applications running on a cloud infrastructure which are accessible through a client interface such as a web browser. Typical examples of this type are Google Docs and Salesforce applications [11, 22]; in the PaaS model, the

consumer can deploy onto the cloud infrastructure applications that he/she created using programming languages and tools supported by the provider. Some of the examples of this type include Google App Engine and Windows Azure [10, 29]; and, lastly in the IaaS model, the provider provisions processing, storage, and other fundamental computing resources[1] where the consumer is able to deploy and run arbitrary software. Amazon EC2 & S3 and Eucalyptus are the examples of this type [2, 7]. The successful implementation of cloud infrastructure requires that both foundational and applied components work together seamlessly.

### 2.2 Role-based support in current IaaS platforms

Role-based security policy [9, 24] has attracted considerable attention in computer security communities over the last two decades, and it has grown to be a proven solution for managing access control in a simple, flexible, and convenient manner. The basic idea behind RBAC is to use the intermediary concept called *role* to provide an indirection mechanism between users and permissions. This indirection mechanism helps reduce errors in user/permission management, support advanced features such as constraints and role hierarchy, and allow for convenient user and permission management schemes based on role-based administration and delegation [1, 3, 26, 30]. RBAC has been successfully implemented in many commercial systems including different flavors of operating systems, database systems, enterprise-based web applications.

Unfortunately, most of the existing IaaS platforms do not support a collection of users associated with their permissions in their user management scheme, and thus there is no support for roles. Each user in the platforms is considered to be independent of others and is provided with a root privilege to access virtualized resources that are requested and serviced on the basis of pay-as-you-go. Therefore, this flat hierarchy rooting from the direct mapping between users and virtualized resources naturally lacks the support for the advanced features that RBAC provides.

Amazon's EC2 [2] is one of the most popular IaaS platforms. Amazon also provides many accompanying services such as storage (S3, ESB) and databases (SimpleDB, Relational Database Service). Interestingly, there is no support for advanced access control mechanisms such as RBAC in these services. The only type of access control we can identify in EC2 is the restriction on operating system (OS) images. A user can upload an OS image and attach an access control list (ACL) to it in order to specify which users can use that image. An image can also be made available to the public. On the other hand, Eucalyptus [7] is another popular IaaS platform, which was started as an academic research and then converted into an open source project. It has been designed to be an exact clone of the Amazon EC2 in terms of functionality. Hence, it is equipped with the same access control mechanism as EC2. However, since it is open source, it is possible to extend the platform to support a variety of advanced features and our approach uses Eucalyptus for a proof-of-concept implementation of our proposed approach.

NASA's IaaS platform called Nebula [15] was initially based on Eucalyptus but it has been rewritten to add its own cloud computing fabric controller called *Nova* to address scalability issues with Eucalyptus. Recently the platform was reconfigured to support the use of roles in its access control mechanism. Based on a short blog posted at [20], the approach taken is twofold: first the frontal controller was connected to an LDAP server for retrieving user and role information, and second a pass and fail gate was implemented on each API call. Though details on this approach have not been published, we suppose that it is similar to our approach from the perspective of supporting roles in access control. However, it has not been known yet how many other features of RBAC have been implemented in this approach. More importantly, there seems to be no support for domain in their approach, which is essentially the main contribution of our approach which allows for domain-based administrative delegation, security, and user/resource management.

Unlike the IaaS platforms we discussed above, Windows Azure [29] is a PaaS type of service platform where users are allowed to develop applications on the Azure AppFabric to deploy them on Microsoft's data centers. The platform also provides storage as well as automatic scaling and load balancing features. Applications deployed within Azure may belong to either or both of *Web* role and *Worker* role. Depending on the role, the application is allowed to perform different tasks. Hence, roles are assigned to the application, not to the user, in this platform.

### 2.3 Role-based support in distributed computing

The usage of roles or other user attributes for access control can be found in various distributed computing systems to provide a finer-grained authorization service. One of them is ISO/IEC's privilege management infrastructure (PMI), based on global namespace, utilizing X.509 attribute certificate framework [5, 8, 13, 25]. PMI is an extension of public key infrastructure (PKI) in the light of authorization. The attribute certificate binds entities to attributes such as roles or groups. On the other hand, SPKI/SDSI [6, 21] is another access control mechanism for distributed systems, based on local namespace, supporting the usage of roles. In the OSF/DCE environment [17, 18], privilege attribute certificate (PAC) that a client can present to an application

[1] Note that networking resource is essential to service computation and storage. Hence, it is included in the computation and storage component.

server for authorization was introduced. PAC provided by a DCE security server contains the principal and associated attribute lists, which are group memberships. The application server works as a reference monitor to make access control decisions based on the comparison between the client's attributes and attributes in ACLs. This approach focused on the traditional group-based access control. Similarly, Thompson et al. [27] developed a certificate-based authorization system called Akenti for managing widely distributed resources.

More recently, the VO Services Project, which is sponsored by US CMS and US ATLAS, has provided a similar attribute-based, fine-grained authorization solution for protecting grid-enabled services and resources [28]. In order to use a grid-enabled resource, a user needs to (1) be issued a grid certificate that represents the user's identity throughout all grid services, (2) extend his/her credential to include membership information of virtual organization (VO)-defined groups and roles, and (3) submit the credential to get access to the resource. The complete solution consists of a suite of software services that help VO and site administrators with user account management, VO authorization management, and authorization policy decisions. Our domain concept is different from VO in that its goal is to represent a single organization or entity as a business user for cloud computing, while VO is a dynamically federated entity encompassing multiple organizations to share and use grid-based resources for collaboration purposes.

## 3 Our approach

In order to support a fine-grained, scalable authorization scheme for IaaS, we discuss a domain-based framework, called *dCloud*, for managing users and virtualized resources in this section. First, we present the formal definitions of the components of the *dCloud* framework. Then, we discuss how to inject role-based access control into the framework by re-defining some of role-based policy constructs.

### 3.1 The *dCloud* framework

The component of IaaS that we are most interested in for our framework is virtualized resources such as virtual machine (VM) with different configuration details, operating system (OS) images, virtual RAM drives, and networking capabilities including public and elastic IP addresses. These kinds of virtualized resource can be found very commonly in existing IaaS platforms, with varying degrees of their abstraction.

There are two types of image that can be provisioned within the *dCloud* framework: service-provided and user-provided. The user-provided images are those that are prepared and uploaded by the service user. For instance, an OS image with pre-configured applications can be created and uploaded to the *dCloud* service provider, so that it can be shared and used by other service users. On the other hand, the service-provided images are those that are provisioned by the *dCloud* service provider[2].

**Definition 1** Let $\mathcal{VR}_p$ and $\mathcal{VR}_u$ denote sets of virtualized resources provided by the service provider and the service user, respectively. The virtualized resource of *dCloud*, denoted as $\mathcal{VR} = \mathcal{VR}_p \times \mathcal{VR}_u$, is represented by n-tuple, where n denotes the number of different kinds of virtualized resource provided as a service by the *dCloud* service provider. For instance, $vr_1 = (VMconfig_1, VMnet_2, VMos_3)$, where $VMconfig_1$ denotes a virtual machine type with 1.7 GB memory, 1 virtual core, 160 GB storage, and 32-bit platform[3]; $VMnet_2$ denotes one public IP address with the port 80 open; and $VMos_3$ denotes a Linux operating system.

The service user can request and access a subset of virtual resources with a root privilege once his/her payment account is set up through an initial setup process such as registration. The initial setup process will also provide the service user with his/her credential, which is required to access the cloud resources; a unique credential can be generated based on either symmetric or asymmetric cryptography such as a secret key and a pair of public/private keys[4].

**Definition 2** Let $\mathcal{U} = \{u_1, \ldots, u_m\}$ denote a set of service users of *dCloud* that consume virtualized resources, and let $\mathcal{C} = \{c_1, \ldots, c_n\}$ denote a set of credentials that consist of keys along with user attributes such as a user identifier. A service user can be uniquely identified by the function $f_{\text{ID}} = \mathcal{C} \rightarrow \mathcal{U}$.

**Definition 3** $\mathcal{UVR} = \mathcal{U} \times \mathcal{VR}$ represents the relation of service user-to-virtualized resource in *dCloud*, and the function $f_{\text{UVR}}: \mathcal{U} \rightarrow 2^{\mathcal{VR}}$ maps a service user to a set of virtual resources.

**Definition 4** Let $\mathcal{AC} = \{ac_1, \ldots, ac_o\}$ denote a set of access rights that the service user has on the virtualized resources. Then, an access control request is defined as a triplet $(u_i, vr_j, ac_k)$ made by a principal, and the function $f_{\text{AUTH}}: \mathcal{U} \times \mathcal{VR} \times \mathcal{AC} \rightarrow \mathcal{D} = \{permit, deny\}$ maps an access control request to an access control decision.

[2]Both types of image are also supported within Amazon EC2.

[3]The similar configuration is called a small instance in Amazon EC2 platform.

[4]Both types are supported as a credential within Amazon EC2 platform.

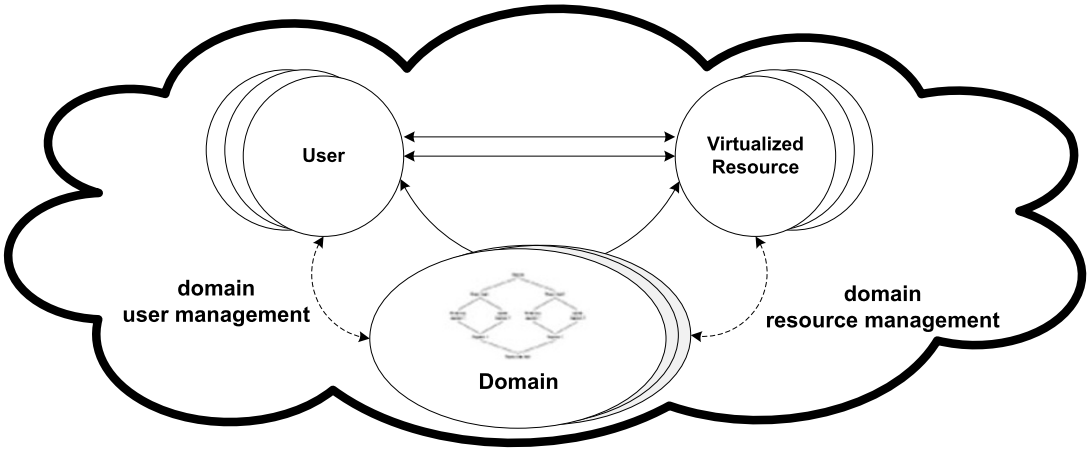


**Fig. 2** Our approach to introducing domain as an intermediary to user-resource direct mapping seen in most of existing IaaS platforms

The three functions defined above, $f_{\text{ID}}$, $f_{\text{UVR}}$, and $f_{\text{AUTH}}$, are closely related to authentication, access control specification, and access control decision, respectively, within the IaaS service provider; and they are usually implemented directly within the provider.

In addition to virtualized resources and service users, we introduce the concept of *domain*, which can provide an indirection mechanism between the service users and virtualized resources. Adding the domain can offer various benefits. First, it can provide a means of decentralized, scalable management of service users and virtualized resources through the delegation of administrative jobs. For example, **SunnyTech.edu** can be delegated the authority to manage its own users and virtualized resources associated with it by **GoCloud.com**. Hence, the function $f_{\text{UVR}}$ whose domain is the set of service users associated with **SunnyTech.edu** can be implemented within the domain of **SunnyTech.edu**. Second, security policies and measures can be applied to the domain level, not the IaaS service provider level. For instance, **SunnyTech.com** can set up procedures and policies to monitor and keep logs of the usage of virtualized resources, and it can implement them on a platform under its authority within **GoCloud.com**. This also means that the functions $f_{\text{ID}}$ and $f_{\text{AUTH}}$ can be locally implemented within the domain of **SunnyTech.edu**. Last, more subscription types can be introduced based on domain-level contracts. Figure 2 depicts the relationship between users, virtualized resources, and domains in our approach.

In our framework, a domain represents an organization with a collection of service users and virtualized resources. Additionally, we assume that the organization should have its own governance and security policies such as discretionary access control (DAC) and mandatory access control (MAC) [12, 14]. We call this an administrative domain, which holds a security repository permitting to authenticate and authorize service users with credentials.

**Definition 5** Let $\mathcal{D} = \{d_1, \ldots, d_p\}$ denote a set of administrative domains in *dCloud*. $\mathcal{UD} = \mathcal{U} \times \mathcal{D}$ represents the relation of service user-to-domain, and the service user may or may not be associated with a domain. Similarly, $\mathcal{VRD} = \mathcal{VR} \times \mathcal{D}$ represents the relation of virtualized resource-to-domain, and the virtualized resource may or may not be associated with a domain.

A single credential identified a service user as either a domain user or a cloud user, but not both; the domain user is associated with a domain, while the cloud user does not belong to any domain. A domain user is uniquely identified by a credential containing the domain information as an attribute through the function $f_{\text{ID}}$. Therefore, for instance, assuming that `Alice` is a faculty member in the computer science department at **SunnyTech.edu** and she has two separate credentials, one issued by her school as a faculty and the other issued by **GoCloud.com** as a regular user, she is identified as a domain user when she tries to access virtualized resources using her school credential. Otherwise, she will be considered to be a cloud user.

### 3.2 RBAC in *dCloud*

In order to show our support for roles within the *dCloud* framework as a construct to formulate, specify, and enforce access control policy, we define our domain as a single RBAC domain where role-based provisioning and management of users and virtualized resources can be designed and implemented[5]. However, it should be noted that a domain can be any type of security administrative domain such as DAC-based or MAC-based. The reason why we chose

[5]Note that the concept of domain is clearly different from the concept of role in our framework in that a domain can contain a set of roles that can be used for access control within the domain.

RBAC is twofold: its policy neutrality and advanced features. Firstly, RBAC is policy neutral so that DAC or MAC policies can be easily expressed using different configurations of RBAC. Further discussions on the policy neutrality is outside the purview of this paper, and we recommend readers interested to refer to [19]. Secondly, RBAC has advanced features such as role hierarchy and constraints. Role hierarchy, defined as a partial order structure, provides a powerful mechanism to ease the administration and management of users and permissions through an inheritance relationship among roles [4, 23]. On the other hand, constraints can be used to represent a higher level of organizational security policies such as separation of duties (SOD). Both role hierarchy and constraints can be added on top of the NIST's core RBAC model to result in the hierarchical and constrained RBAC model, respectively. In this paper, we use the hierarchical RBAC model for *dCloud*.

**Definition 6** The RBAC components supported within a single domain $i$ are as follows.

- $\mathcal{U}^{d_i} \subset \mathcal{U}$ represents the set of domain users.
- $\mathcal{P}^{d_i} = \mathcal{VR}^{d_i} \times \mathcal{AC}$ represents the set of permissions to use virtualized resources, where $\mathcal{VR}^{d_i}$ represents a set of virtualized resources associated with the domain $i$.
- $\mathcal{R}^{d_i}$ represents the set of roles within the domain $i$.
- $\mathcal{S}$ represents the set of sessions.
- $\mathcal{UA}^{d_i}$, $\mathcal{PA}^{d_i}$, and $\mathcal{RH}^{d_i}$, representing the relation of user-to-role assignment, permission-to-role assignment, and role hierarchy, respectively. $\mathcal{RH}^{d_i}$ is partial order on $\mathcal{R}^{d_i}$, written as $\preceq$.
- user: $\mathcal{S} \rightarrow \mathcal{U}^{d_i}$ represents a function mapping each session $\mathsf{s}_j$ to the single domain user.
- roles: $\mathcal{S} \rightarrow 2^{\mathcal{RH}^{d_i}}$ represents a function mapping $\mathsf{s}_j$ to a set of roles, where $\mathsf{roles} \subseteq \{\mathsf{r} | (\exists \mathsf{r}' \succeq \mathsf{r})[(\mathsf{user}(\mathsf{s}_j), \mathsf{r}') \in \mathcal{UA}^{d_i}]\}$ and $\mathsf{s}_j$ has permissions $\bigcup_{\mathsf{r} \in \mathsf{roles}(\mathsf{s}_j)} \{\mathsf{p} | (\exists \mathsf{r}'' \preceq \mathsf{r})[(\mathsf{p}, \mathsf{r}'') \in \mathcal{PA}^{d_i}]\}$.

The components above subsume the components from the core RBAC model [24] with some of them redefined. The semantics and usage of each of the components are not much different from those of the traditional RBAC components. However, we believe that it is necessary to highlight some of the components from the perspective of cloud computing which has some restrictions as well as advantages. First, unlike the traditional RBAC system where policy data are stored within the organizational security perimeters, they are stored in clouds where the policy data could be considered to be more vulnerable. Hence, the security and privacy of the data are important issues to consider when supporting RBAC for *dCloud*. Depending on the trustworthiness of the *dCloud* service provider, different security mechanisms such as encryption of policy data can be considered, and we will discuss more in detail in the next section where we discuss a system architecture based on our framework.

Second, the semantics of roles can be expanded so that they could be understood and used in an inter-domain relationship, including one between the *dCloud* service provider and a domain. There has been much work on the importance and usefulness of inter-domain role mapping and its implications on entitlements and revocation in literature, so this issue should be considered in supporting RBAC for *dCloud*. For instance, assuming the *dCloud* service provider wants to provide all service users with basic permissions to access limited resources, role mapping between the service provider and domains could be considered so that the inheritance of this basic set of permissions can be made to domain roles. In other words, permissions are inherited through a role hierarchy encompassing the cloud roles and domain roles.

## 4 System design

The complexity of authorization services comes from access control policy specification, decision, and enforcement. We believe that our domain-based framework leads to a more scalable authorization service in this regard by maximizing security policy autonomy through the delegation of complex works relating to security policy specification and decision from the cloud service provider; our framework basically increases the number of policy specification and decision points through decentralization to scale for a large number of users and authorization requests from them.

The security architecture we propose based on our framework is slightly modified from a traditional XML-based security architecture with multiple points associated with access control policy storage, administration, decision, and enforcement. In addition to being scalable, it is also extensible because of its simple, component-based approach. As shown in Fig. 3, the *dCloud* service provider is composed of seven components, three of which are directly related to the policy administration, storage, and decision of each domain, and four of which are components necessary for applying policy-based security enforcement within the *dCloud* service provider. Note that even though the policy decision can be made at either domain-level for the domain user or provider-level for the cloud user, the security policy can be enforced only at the service provider. The descriptions of the seven components are as follows.

- *Domain Policy Administration Point (D-PAP)*: This component contains the capability to perform several different tasks within a domain, guaranteeing the security and privacy of information stored and exchanged. These tasks relate to user account administration and role-based policy management.

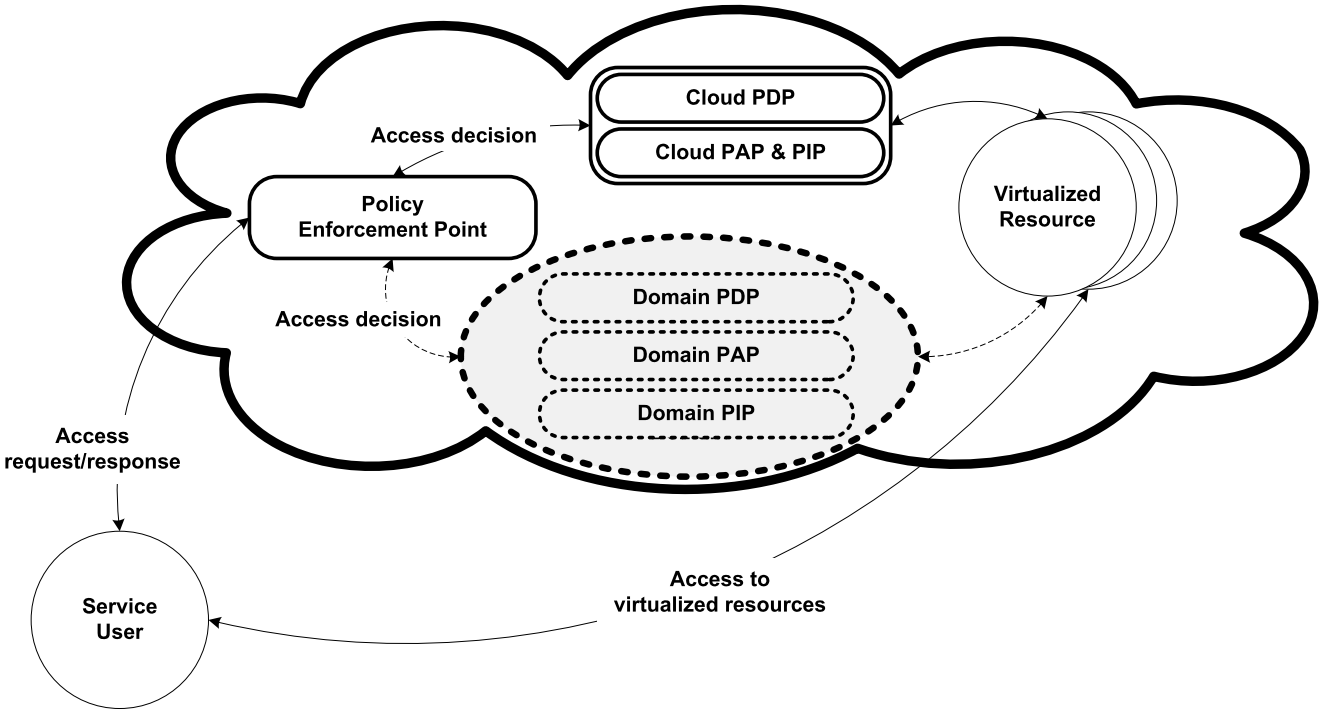


**Fig. 3** Policy-based security architecture based on the *dCloud* framework

– *Domain Policy Information Point (D-PIP)*: This component pertains to a system that acts as a source of security policies within a security administrative domain. Security policies are manipulated by the D-PAP.
– *Domain Policy Decision Point (D-PDP)*: This component works as a system that evaluates various types of applicable security policy from D-PIP and makes an appropriate decision upon some actions from the policy enforcement point (PEP).
– *Policy Enforcement Point (PEP)*: This component works as a system that performs various types of security enforcement, based on the decision made by either D-PDP or Cloud PDP upon its request.

The functionalities of the remaining three components, Cloud PAP, Cloud PIP, and Cloud PDP, are almost the same as D-PAP, D-PIP, and D-PDP, respectively, with the only difference among them being where their policy can be applied, i.e., either domain-level or cloud-level.

Auditing works as a crucial role for any type of security system, and our system is no exception to that. However, our architecture, shown in Fig. 3, does not contain any component related to auditing. This is mainly because we want to focus more on showing how security policy is used within our *dCloud* framework, and also because since the auditing component is likely to involve all interactions among the seven components, we assume that each domain has a component in charge of auditing the activities among its PAP, PIP, PDP, and PEP. Likewise, we also assume that the *dCloud* service provider has an auditing component logging the activities among its PAP, PIP, PDP, PEP, and the PDP of each domain.

## 5 System implementation

We discuss our approach to implementing the system architecture based on the *dCloud* framework in this section. The open source version of Eucalyptus [7] has been modified and extended to show the feasibility of our approach. As previously discussed, Eucalyptus has no notion of roles or domains to support its access control mechanism. We first implemented the objects representing domains and other role-based policy constructs such as roles and permissions within Eucalyptus. We then modified the web interfaces of Eucalyptus so that they can be used to administer domain-based security policies as well as provider-level security policies. These security policies are stored using `Hibernate`, which is the default tool for persistent data storage and management within Eucalyptus. A reference monitor working as a policy decision point was also implemented, and finally `Euca2ools` were modified to leverage the reference monitor for authorization.

In order to give a better picture of what we have implemented, we believe that it is necessary to depict the general architecture of Eucalyptus. We then present our implementation efforts in detail as well as the performance analysis in subsequence sections.

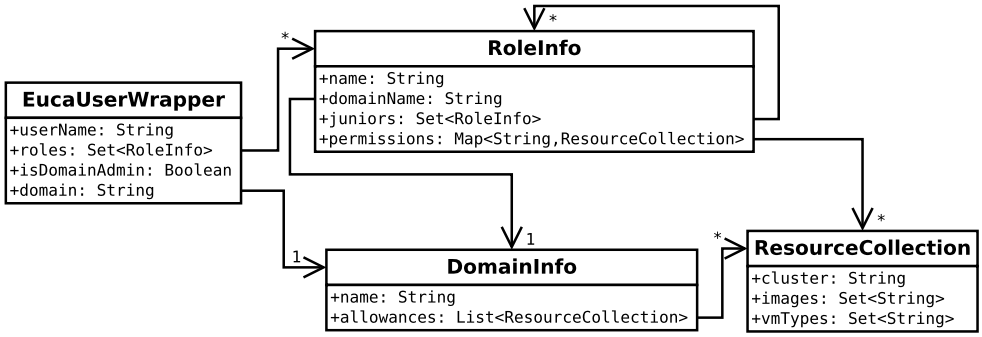


**Fig. 4** Object design to support domains and RBAC within Eucalyptus

### 5.1 The architecture of eucalyptus

Eucalyptus is organized into five components, each of which is responsible for operation of a different part of the platform as follows:

1. Cloud Controller (CLC): This is the main component which governs the system and exposes a query interface for users to communicate with the system. It also leverages a web interface for administration, user registration and retrieving user credentials. Our extension effort is mainly made at this component.
2. Walrus: This is the component equivalent to Amazon S3 for Eucalyptus.
3. Cluster Controller (CC): Each cluster has a CC responsible for resource allocation among the nodes within that cluster.
4. Storage Controller (SC): This is the component equivalent to Amazon ESB for Eucalyptus. Each cluster has a SC that exports block storage devices over the LAN for VMs to mount them.
5. Node Controller (NC): This is responsible for running/terminating virtual machines and networking of them.

In the open source distribution, the CLC is composed of a number of tightly coupled modules. They are all compiled and packaged into separate jar files and loaded by the same classloader during the boot process of the system. For example, the web administration module is compiled into the package called **eucalyptus-www.jar** and loaded/started by the bootstrapper. We have implemented a separate module for the domain and RBAC called **rbac-manager.jar** and put all the codes related to roles and domains in this package.

### 5.2 Injecting domains and roles into Eucalyptus

In Eucalyptus, all users are treated the same. By introducing the concept of domains to Eucalyptus, we can group users and manage access control administration for a domain separately from other domains. Our low-level additions to the Eucalyptus platform are several entities that represent users, roles, permissions, domains, and a reference monitor together with its accompanying tools. Entities we implemented and relationships between them are shown in Fig. 4. **EucaUserWrapper** object wraps the original entity that represents the users in the system and adds it attributes related to the *dCloud* such as role and domain information. **ResourceCollection** objects are abstractions to any set of virtual resources that is assignable to a role or a domain but not directly to a user. For example, a user who possesses the role associated with the following **ResourceCollection** object can only create a virtual machine of type *m1.medium*, only in the cluster named *ZoneA*, and can use either of the images listed:

```
cluster=``ZoneA'',
images=[emi-AAAAAA,eri-BBBBBB],
vmTypes=[m1.medium]
```

Each domain in the system is represented by a **DomainInfo** object which has one-to-many relationship with the **ResourceCollection** object. This relationship enables the cloud administrator to create domains and assign virtualized resource allowances to them only to be reassigned to roles of that domain by a domain administrator. Lastly, each **RoleInfo** object represents a role in the system and contains information regarding the domain it belongs to as well its junior roles. It has a many-to-many relationship with itself facilitating a role hierarchy.

After having implemented objects representing domains and role-based policy constructs within Eucalyptus, we modified its web interface module so that the cloud administrator can manage domains and virtualized resources to be assign/de-assigned to the domains, as illustrated in Fig. 5. The left snapshot shows that a new domain called **CS-Dept** is to be created, while the right snapshot depicts the process to manage virtualized resources assigned to the domain. We also developed a web-based policy administration point for domains, as shown in Fig. 6. Using the web interface, the domain administrator can manage user-to-role, permission-to-role, and role hierarchy relations. The left snapshot shows that a new role called **Student** is to be created with a set of permissions, and the **Student** role has one junior role called **CloudUser**, which is associated with basic cloud provider-level permissions. The right snapshot shows that a new role

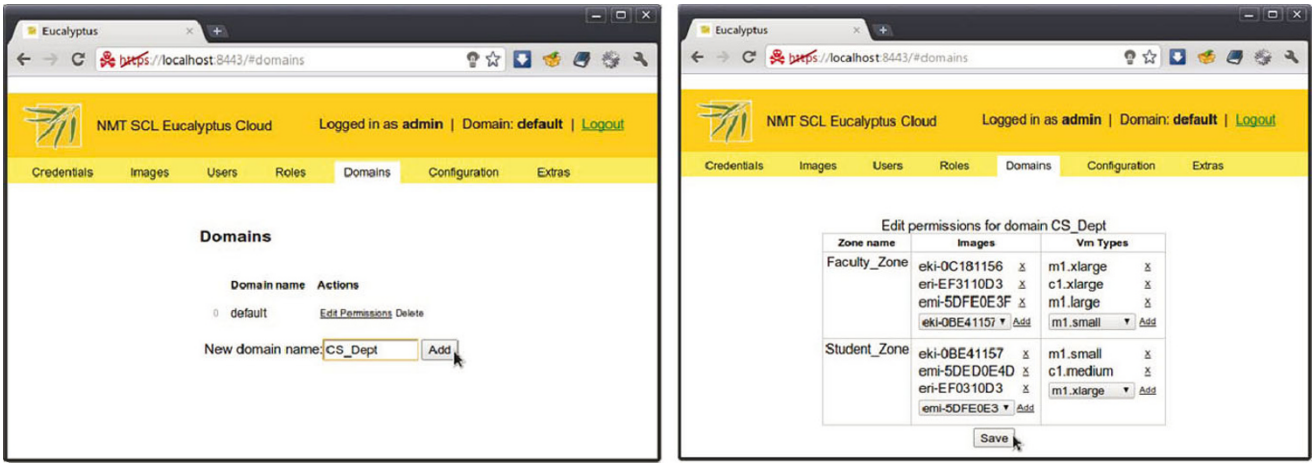


**Fig. 5** The web interface for managing domains and virtualized resources. The default web interface of Eucalyptus was modified to have additional tab called domains to support the capability of domain and virtualized resource management

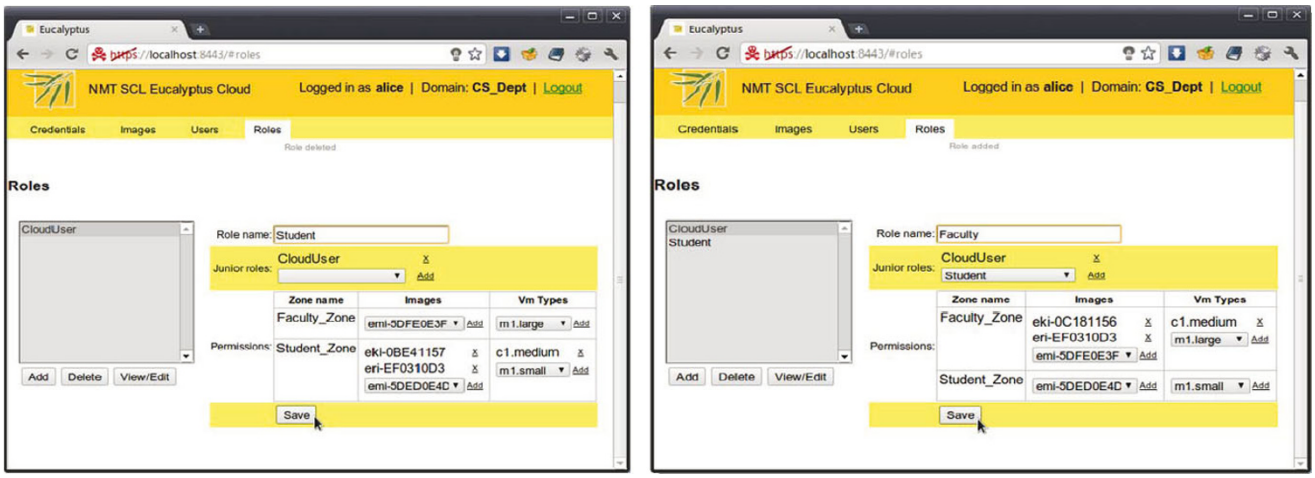


**Fig. 6** The web interface for managing roles and permissions. The default web interface of Eucalyptus was modified to have additional tab called roles to support the capability of user-to-role, role-to-permission, and role hierarchy management

called **Faculty** is to be created with a set of permissions, and it has the **Student** role as well as the **CloudUser** role as its junior roles, so that it can inherit permissions assigned to the junior roles. In addition to the basic information of a user, the **Users** tab allows the domain administrator to view all roles that the user is assigned.

The reference monitor, working as a policy decision as well as enforcement points, is involved only in processing a VM creation or termination request. The request message goes through several components in which the validity and acceptability of the parameters are verified within Eucalyptus. The internal messaging between those components is done via Mule enterprise service bus (ESB) that also includes the processing of such requests. We have configured the Mule so that a VM creation or termination request message is passed through the reference monitor as the last stage of the validation process. Mule configuration was done in compile-time with XML files in which services and associated endpoints are configured. We also defined a new endpoint called **RolesVerifyWS** and a new service called **RolesVerify** in

```
clc/modules/.../eucalyptus-services.xml,
clc/modules/.../eucalyptus-verification.xml,
```

respectively. The reference monitor receives a **VmAllocationInfo** object which contains the necessary information to run VMs such as the user ID, requested kernel images, and VM type. It first retrieves the **UserInfo** object associated with the requesting user and retrieves user's roles as a list. Starting from these roles, it initiates a breadth-first-search over the role hierarchy to find all of the image, VM type, and cluster permissions. During the search, if all of the required permissions are found in one or more roles, the search is terminated and the reference monitor allows message to pass through. If the search terminates with consuming all of

the role hierarchy and not finding all of the required permissions, an exception is thrown which will prevent the request from further processing and return the exception message to the client. The following shows the details of the decision process.

**Procedure:** AuthEval(*VmAllocInfo*):
1: *User* ← *VmAllocInfo*.User /*Get UserInfo*/
2: *ReqResources* ← *ReqResourcesList*(*VmAllocInfo*) /*Create a list of requested resources*/
3: *Queue* ←<> /*Create an empty queue: unchecked roles*/
4: *Seen* ← {} /*Create an empty set: checked roles*/
5: **for all** *Role* ∈ *User*.Roles **do**
6: *Queue*.Enqueue(*Role*)
7: **while** *Queue* is not ∅ **do**
8: *Role* ← *Queue*.Poll() /*Remove the head of the queue*/
9: Check(*Role*, *ReqResources*)
10: **if** *ReqResources* is ∅ **then**
11: **return** true
12: **for all** *JuniorRole* ∈ *Role*.JuniorRoles **do**
13: **if** *JuniorRole* ∉ *Queue* and *JuniorRole* ∉ *Seen* **then**
14: *Queue*.Enqueue(*JuniorRole*)
15: *Seen*.Add(*Role*)
16: **return** false

**Procedure:** ReqResourcesList(*VmAllocInfo*):
1: *RRList* ← [] /*Create an empty list*/
2: *RRList*.Add(*VmAllocInfo*.Cluster)
3: *RRList*.Add(*VmAllocInfo*.VmType)
4: *RRList*.Add(*VmAllocInfo*.ImageId)
5: *RRList*.Add(*VmAllocInfo*.KernelId)
6: *RRList*.Add(*VmAllocInfo*.RamdiskId)
7: **return** *RRList*

**Procedure:** Check(*Role*, *ReqResources*):
1: **for all** *Resource* ∈ *ReqResources* **do**
2: **if** Permits(*Role*, *Resource*) **then**
3: *ReqResources*.Remove(*Resource*)

## 5.3 Performance analysis

We conducted an experimentation to study the effect of adding domains to Eucalyptus on its performance, especially on the time taken for access control decision made during the processing of a VM creation request. The independent variables that we were interested in for our experimentation are the support for domain, the number of domains supported, and the number of concurrent access requests. Other possible variables closely related to RBAC such as the height of role hierarchy was ignored since the effect of them has been previously studied in literature.We first measured the response time of the original Eucalyptus for VM creation requests with changing variables such as the number of users, clusters, and images. Then we measured the same for a revised version of Eucalyptus, called Sclyptus, for the comparison purpose.

### 5.3.1 Testbed setup

In order to setup a testbed, we used two different types of machine: **Type A**—Dell PowerEdge 1900 server (3.0 GHz, 8 cores, 8 GB RAM, 2 × 500 GB, 1 GBit Ethernet) running Eucalyptus and Sclyptus, and **Type B**—IBM IntelliStation (4.3 GHz, 1 GB RAM, 40 GB, 100 Mbit Ethernet) running concurrent access request launchers implemented in Python[6]. Two type A servers and ten type B PCs were used for this experimentation.

### 5.3.2 Procedure

The method we used made continuous requests to the servers until the standard deviation of the response times for all requests became less than 1%. During each test, there were 1000 threads each making a request to the cloud controller (CLC) and calculating the elapsed time when the response is received. These threads were distributed among 10 type B machines and at each machine the "driver" thread collected the measured response times, calculated the average and the standard deviation.

In order to be able to make requests that will be granted for access, the driver tool needs to know the domain/role/permission/cluster/user configuration uploaded to the Sclyptus. To achieve this, we created another tool that reads the image names in the system and creates a random configuration based on some parameters such as number of domains, number of clusters, number of roles and the role hierarchy, image permissions per role etc. After creating a random configuration, this tool applies that particular configuration to the Sclyptus by using the CLI tools that we developed to automate the process. The driver reads this configuration and builds a queue of random tuples of (user, cluster, machine image, kernel image, ramdisk image) in a "producer" thread. Each request thread acts as a "consumer" and pops a tuple from the queue before making a new request.

### 5.3.3 Results

Figure 7 shows the response time measured for Eucalyptus. Each bar in the figure represents a configuration tuple (number of users, number of clusters, number of images) in the system. We tested Eucalyptus with variables ranging as follows: *Users* (10,100,1000), *Clusters* (1,5,10), and *Images* (10,100,250,1000). For the stock Eucalyptus (version 2.0), response times were ranging from 0.087 seconds to 0.105 seconds. That means, Eucalyptus was able to respond

[6]Note that no machine was used to run the node controller (NC), since we did not need to run actual VM instances. NCs were disabled on purpose so that the response time we measured excluded VM creation process overheads, which could invalidate our performance analysis.

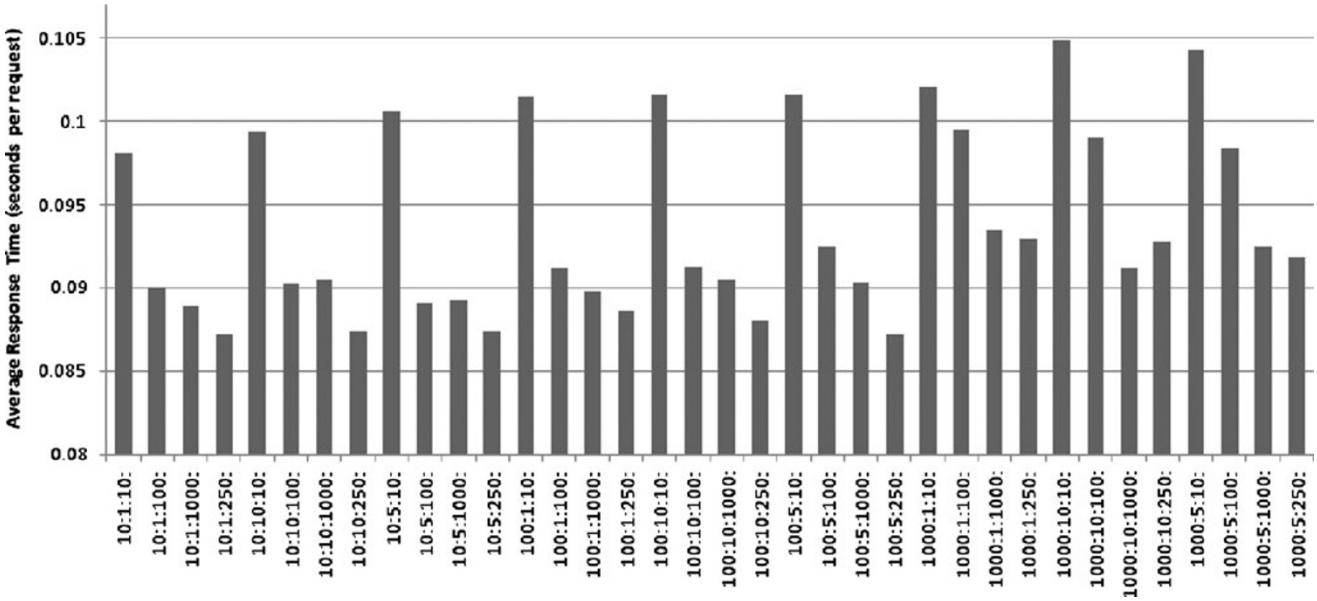


**Fig. 7** Performance test result for Eucalyptus. Each label on the *horizontal axis* depicts a configuration tuple in the form of (number of users, number of clusters, number of images)

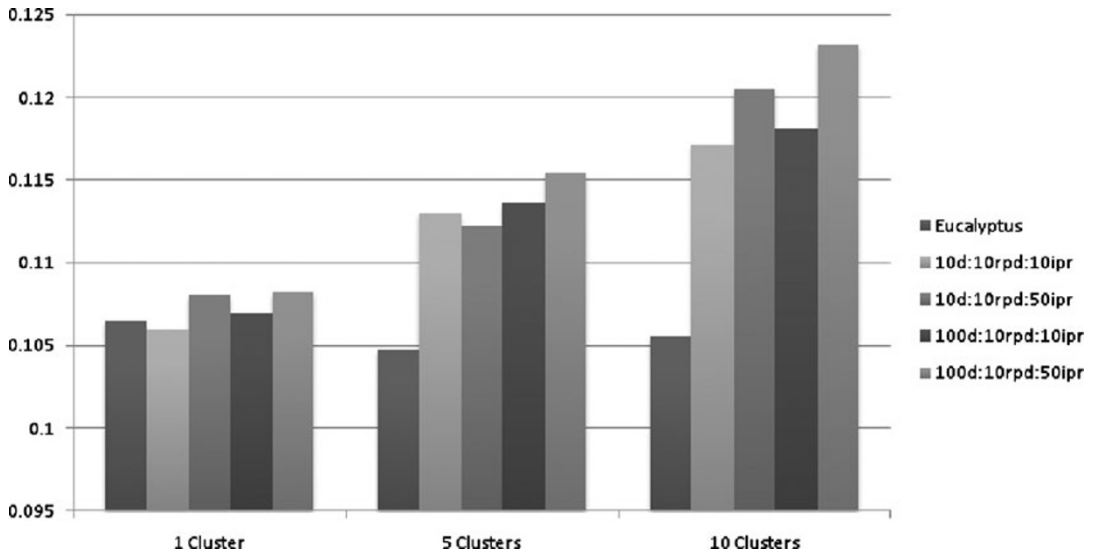


**Fig. 8** Performance test result from Sclyptus. 100d:10rpd:50ipr means 100 domains 10 roles per domain, 50 images per cluster per role

to each request in at most ∼105 seconds on average, or responded to 60/0.105 ≈ 570 requests per minute when there were 1000 outstanding requests.

To compare the response times of Eucalyptus, we tested our Sclyptus with fixed amount of users (100), images (1000), and roles per domain (10 with a hierarchy, and all users were assigned to the most senior role in their domain) in the system and changed the following parameters: *Domains* (10,100), *Clusters* (1,5,10), and *Image permissions per cluster per role* (10,50). Each role was given permissions to use 10 or 50 images in each cluster. As shown in Fig. 8, Sclyptus performs very well and adds little overhead even under heavy load. Comparing Eucalyptus and Sclyptus configuration "100d:10rpd:50ipr", the average response time for a request only increased from 0.105 to 0.123. That means, the throughput of the system was lowered from 60/0.105 ≈ 571 requests per minute to 6 /0.123 ≈ 487 requests per minute which results into 15% performance loss. Considering that the number of images per role would not be more than a few in real world and there are usually only a few clusters per cloud provider, we believe that the domain and RBAC support is a viable extension to IaaS platforms.

## 6 Conclusion and future work

In this paper we discussed a domain-based framework called *dCloud* for efficiently managing users and virtualized resources in cloud computing. Specifically, our framework introduces an additional layer called domain to the *user-resource* direct approach taken by most of existing infrastructure as a service (IaaS) service providers in order to support organization-based policy support, flexible authorization management, and better pricing for business users. We also presented how to design and implement a proof-of-concept prototype by modifying an existing IaaS platform called Eucalyptus to show the feasibility of our approach. Finally, we discussed the performance analysis based on our prototype implementation.

Our work is by no means complete. Our immediate future work includes support of additional operations on virtualized resources in order to fully benefit from advanced access control models such as RBAC and study their security and performance implications. In order to do that, we will enhance our current implementation to include authorization support for Walrus, which provides storage service like Amazon S3. We also study how to support for different types of policy database for the PIP for domains. This will include the support for the relational database as well as directory service. In doing so, we will also investigate how a virtual directory service can be utilized within our security architecture. Virtual directories, also known as virtual directory services (VDS), are structures designed to handle the complicated task of combining data from multiple data sources into a single data source, which is presented to end users. VDS are also able to selectively present specific information from a data source. The use of VDS for the *dCloud* service provider seems to be beneficial in many aspects. Finally, we will study the applicability and consequence of supporting the concept of virtual organization (VO) in our framework. This will include the potential use of attribute-based credentials and the trust relationship between VOs.

**Acknowledgements** This work was partially supported at the Secure Computing Laboratory at New Mexico Tech by the grant from the National Science Foundation (NSF-IIS-0916875).

## References

1. Ahn G-J, Sandhu R (1999) The RSL99 language for role-based separation of duty constraints. In: Proceedings of 4th ACM workshop on role-based access control, pp 43–54, Fairfax, VA, 28–29 October 1999. ACM, New York
2. Amazon Elastic Compute Cloud and Simple Storage Service. http://aws.amazon.com
3. Barka ES, Sandhu RS (2000) Framework for role-based delegation models. In: Proceedings of 16th annual computer security application conference, New Orleans, LA, December 2000
4. Crampton J (2003) On permissions, inheritance and role hierarchies. In: Proceedings of 10th ACM conference on computer and communication security, Washington, DC, October 2003
5. Dimmock N, Belokosztolszki A, Eyers D, Bacon J, Moody K (2004) Using trust and risk in role-based access control policies. In: Proceedings of 9th ACM symposium on access control models and technologies, Yorktown, NY, June 2004
6. Ellison C, Frantz B, Lampson B, Rivest R, Thomas B, Ylonen T (1999) SPKI certificate theory. RFC 2693, September 1999
7. Eucalyptus Open Source. http://open.eucalyptus.com/
8. Farrell S, Housley R (2001) An Internet attribute certificate profile for authorization. Technical report, PKIX Working Group, June 2001
9. Ferraiolo DF, Sandhu R, Gavrila S, Kuhn DR, Chandramouli R (2001) Proposed NIST standard for role-based access control. ACM Trans Inf Syst Secur 4(3)
10. Google Apps. http://www.google.com/a
11. Google Doc. http://docs.google.com/
12. Harrison MH, Ruzzo WL, Ullman JD (1976) Protection in operating systems. Commun ACM 19(8):461–471
13. ITU (2000) ITU-T recommendation X.509. Information technology: open systems interconnection—the directory: public-key and attribute certificate frameworks. ISO/IEC 9594-8
14. McLean J (1985) A comment on the 'Basic security theorem' of Bell and LaPadula. Inf Process Lett 20(2):67–70
15. NEBULA: NASA's Cloud Computing Platform. http://nebula.nasa.gov/
16. NIST. (2009) Nist working definition of cloud computing. Technical report. http://csrc.nist.gov/groups/SNS/cloud-computing/index.html
17. Open Software Foundation (1992) OSF DCE 1.0 application development guide. Cambridge, MA
18. Open Software Foundation (1992) OSF DCE 1.0 introduction to DCE. Cambridge, MA
19. Osborn S, Sandhu R, Munawer Q (2000) Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Trans Inf Syst Secur 3
20. RBAC support for Nebula. http://nebula.nasa.gov/blog/2010/jun/nebulas-implementation-of-role-based-access-contro/
21. Rivest RL, Lampson B (1996) SDSI—a simple distributed security infrastructure. Technical report, September 1996
22. Salesforce's the Sales Cloud. http://www.salesforce.com/crm/sales-force-automation/
23. Sandhu R, Munawer Q (1998) The RRA97 model for role-based administration of role hierarchies. In: Proceedings of 14th annual computer security application conference, pp 39–49, Scotsdale, AZ, 7–11 December 1998
24. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE (1996) Role-based access control models. IEEE Comput. 29(2):38–47
25. Shin D, Ahn G-J, Cho S (2002) Role-based EAM using x.509 attribute certificate. In: Proceedings of sixteenth annual IFIP WG 11.3 working conference on data and application security, Cambridge, UK, 29–31 July 2002
26. Shin D, Ahn G-J, Cho S, Jin S (2003) On modeling system-centric information for role engineering. In: Proceedings of 8th ACM symposium on access control models and technologies, Como, Italy, 2–3 June 2003
27. Thompson M, Johnston W, Mudumbai S, Hoo G, Jackson K, Essiari A (1999) Certificate-based access control for widely distributed resources. In: Proceedings of 8th USENIX security symposium, Washington, DC, 23–26 August 1999
28. VO Services Project by US CMS and US ATLAS. http://www.fnal.gov/docs/products/voprivilege/
29. Windows Azure. http://www.microsoft.com/azure/
30. Zhang L, Ahn G-J, Chu B (2001) A rule-based framework for role-based delegation. In: Proceedings of 6th ACM symposium on access control models and technologies, pp 153–162, Chantilly, VA, 3–4 May 2001