Towards a Multi-tenancy Authorization System for Cloud Services

Jose M. Alcaraz Calero[†*], Nigel Edwards[*], Johannes Kirschnick[*], Lawrence Wilcock[*], Mike Wray[*]
[*]Automated Infrastructure Laboratories
Hewlett Packard Laboratories
BS34 8QZ Bristol
United Kingdom
Email: {nigel.edwards,johannes.kirschnick,lawrence_wilcock,mike.wray}@hp.com
[†]Communications and Information Engineering Department
University of Murcia
Computer Science Faculty
Murcia, Spain, 30100
Email: jmalcaraz@um.es

***Abstract*—Cloud computing presents new security challenges to control access to information in cloud services. This paper describes an authorization model suitable for cloud computing in which hierarchical role-based access control, path-based object hierarchies and federation are supported. Moreover, it also proposes the architecture of an authorization system which implements the model. This architecture has been implemented and some technical implementation details together with performance results from the prototype are provided. Security, privacy and trust management aspects for the authorization system are also described.**

I. Introduction

Cloud computing [1] is an emerging computing technology which allows businesses to implement their own business services using on-demand IT infrastructures. This idea behind this approach is to provide a new model of infrastructure provisioning on which business can create elastic on-demand IT infrastructures according to their ever changing requirements. These on-demand infrastructures may enable end users to use the business services without installation and access them at any computer with internet access. These services are so-called cloud services.

To achieve this end, the cloud computing defines a stack composed of three well-known layers [2]: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (Saas). Firstly, the *IaaS* layer uses the physical storage space and processing capabilities to provide as a service to deploy on-demand virtual IT infrastructures. Secondly, the *PaaS* layer uses the previous layer to provide middleware services that are used to implement the final cloud services. This layer may contain middleware services such as security services, user management services, distributed processing services, etcetera. The services provided in this layer are usually multi-tenancy services. Multi-tenancy support is defined as the capability of a single software instance to provide its service to several parties simultaneously. Finally, the *SaaS* layer uses the previous two layers to offer the cloud services to the end users.

This paper describes a multi-tenancy authorization system suitable to be a middleware service in the *PaaS* layer. This authorization system provides access control to the information and services of all the different cloud services which are using the cloud infrastructure. Each business may provide several cloud services and these services could collaborate with other services belonging either to the same organization or to a different organization and the authorization system has to support these collaboration agreements - federation.

This paper is structured as follows: section II describes related work on authorization systems for the cloud computing. Our authorization model is explained in section III. Next, section IV describes the architecture of the authorization system suitable for cloud computing environments which implements the model. Section V provides some implementation details and section VI some performance results. Finally, section VII provides some conclusions and discusses future work.

II. Related Work

This section describes related work on authorization models and systems related to the cloud computing. Chow et al [3] characterize the problems and their impact on the adoption of cloud computing as a way of outsourcing computation without outsourcing control. The authors remark that traditional enterprise authentication and authorization frameworks do not naturally extend into the cloud. They claim that traditional frameworks are not able to deal with multiple cloud resources and integrate cloud security data into their security metrics and policies. In addition, Sangroya et al [4] describe the security mechanisms that are used by major cloud

service providers and propose a risk analysis framework that can be used to analyse the data security risks before putting confidential data into a cloud computing environment. The above research illustrates the need for new authorization services and models suitable for the cloud computing to provide access control in a distributed, multi-tentanted, federated environment.

Euycalyptus [5] is an open-source cloud computing system. This system provides an authorization system to control the execution of the virtual machines (VMs) that compose the cloud infrastructure. This authorization system protects running VMs to ensure that only administrators and the owner of these VMs are able to access them. This authorization system is focused only on the control of access to the VMs and it does not provide any capability to protect the resources of the cloud services running inside the VMs.

The remainder of this section describes alternative authorization model proposals. What distinguishes our work is that it provides federated path-based access control. We believe the abstraction of a path is better suited to authorization in cloud systems than alternatives based on labels or tickets. The abstraction of a path is often used by programmers, particularly in REST [6] based web-services that are often seen in cloud systems. We believe it is important to match the abstraction of the authorization system to the abstractions that are used by programers. This is important, because it is the programmers implementing the services who are responsible for ensuring the services enforce the security policy.

Berger et al [7] promote an authorization model based on both role-based access control and security labels. The security labels represent colours and these colours are used to control the access to data sharing, to virtual machines, network resources, etcetera. Zhang et al [8] promote an authorization scheme for the cloud computing based on session keys shared between the cloud services and the users in order to be able to access to the protected resources. We believe path-based authorization is a more natural fit to the abstractions used by programmers of cloud applications and therefore easier to use than either of these approaches.

Danwei et al [9] provides an access control architecture based on the UCON [10] authorization model. This model manages concepts such as subject, object, right, obligation, condition, attribute, etcetera. The authors propose an authorization architecture based on UCON suitable for the cloud computing. The main contribution of this proposal is the inclusion of a negotiation model in the authorization architecture to enhance flexibility of the access control on the cloud services. Then, when the access requested mismatches with access rules, it allows users to get a second access choice through negotiation in certain circumstances, instead of refusing access directly. This work is still at the conceptual stage, its usability in a cloud environment has yet to be demonstrated.

Hu et al [11] present a new semantic access control policy language (SACPL) for describing access control policies in cloud computing environments. They propose an ontology-based access control to tackle the interoperability issue of distributed access control policies. The focus of this work is interoperability between different schemes and models. We present a specific model and implementation of that model.

Di Capitani et al [12] and Yu et al [13] present novel solutions for the enforcement of access control and the management of its evolution in cloud environments. The former applies selective encryption as a method to enforce authorizations whereas the latter enforces access policies based on data attributes. These researches can be used to complement the authorization model proposed by controlling the enforcement of the access control information on the untrusted cloud servers.

III. Authorization Model

This section describes an authorization model for controlling access to resources in a cloud system. In essence, an authorization model has to determine if a *subject* has the *privilege* to perform a given action over the controlled *object*. This fact can be represented by a 3-tuple (*Subject*, *Privilege*, *Object*). However, this 3-tuple has to be extended to enable multi-tenancy support in which different *Issuers* are using simultaneously the authorization system. Thus, a 4-tuple (*Issuer*, *Subject*, *Privilege*, *Object*) can be defined. Moreover, the authorization model can be extended to protect different types of *object*: the previous 4-tuple can be extended in the 5-tuple (*Issuer*, *Subject*, *Privilege*, *Interface*, *Object*) where *Interface* represents the way in which the *Object* can be interpreted. Then, this 5-tuple may be interpreted as: the *Issuer* says that the *subject* has the *privilege* to perform a given action over the object *object* associated to the type *Interface*. For example, (*Jose*, *Nigel*, *Read*, *CloudStorage*, *\root*) may be interpreted as: *Jose* says that *Nigel* can *Read* the *\root* folder associated to the *CloudStorage* service. Note that a privilege may enable multiple actions. For example *Write* might enable the actions *delete* and *update*.

To support the authorization model a role-based access control (RBAC) [14] is incorporated. The authorization model relies on the usage of *Roles* as a set of privileges that could be assigned to a user. Thus, the membership of a user to a given role can be defined by means a 3-tuple (*Issuer*, *User*, *roleName*) which may be interpreted as: the *Issuer* says that the *User* has the given *roleName*. *Role* is scoped by the issuing party, so we write $role(Issuer, roleName)$ to represent a *Role*. Hence $role(Jose, Admin)$ and $role(Nigel, Admin)$ are different. In the first *Jose* is issuing the *Admin* roleName, in the second *Nigel* is issuing the *Admin* roleName. An authorization service may treat these two *Roles* differently. Any privileges granted to a *Role* is granted to the *Users* with that role. There is a transitiveness relationship between *Role* and *User* and it has to be inferred in the authorization

system. For example, (*Jose*, *Nigel*, *DatabaseAdmin*) can be interpreted as *Jose* says that user *Nigel* belongs to the *DatabaseAdmin* role. The previous 5-tuple is extended to define privileges over *Roles*. Thus, the *Subject* is defined as either *User* or *Role*. The result is a new 5-tuple defined as (*Issuer*, [*User*|*Role*], *Privilege*, *Interface*, *Object*). In order to avoid any ambiguity, the *User* and *Role* are now represented as *type*(*value*) in this 5-tuple. For example *user*(*Nigel*) and *role*(*Jose*, *DatabaseAdmin*) respectively. Then, the following example (*Jose*, *role*(*Nigel*, *Admin*), *Read*, *CloudStorage*, *root*\) may be interpreted as *Jose* says that role *role*(*Nigel*, *Admin*) can *Read* the folder *root*\ of the *CloudStorage*.

Additionally, hierarchical role based access control (hRBAC) [15], which enables a hierarchical subject grouping, has been incorporated to the authorization model. This kind of access control enables the definition of hierarchies of *Roles*. A *Role* can be defined as a specialization of another more generic *Role*. For example, *role*(*Jose*, *DatabaseAdmin*) could be defined as a specialization of *role*(*nigel*, *Admin*), and therefore, all the privileges defined over *role*(*Nigel*, *Admin*) are also granted to *role*(*Jose*, *DatabaseAdmin*). To this end, the previous 3-tuple has been extended as (*Issuer*, [*User*|*Role*], *Role*). This enables us to define *Roles* belonging to another *Role*. Again, to avoid any ambiguity, the *User* and *Role* are defined following this pattern: *type*(*value*). For example, (*Nigel*, *role*(*Jose*, *DatabaseAdmin*), *Admin*) can be interpreted as *Nigel* says that *role*(*Jose*, *DatabaseAdmin*) is a sub-role of *role*(*Nigel*, *Admin*): all privileges granted to *role*(*Nigel*, *Admin*) will also be granted to *role*(*Jose*, *DatabaseAdmin*).

Both *User* and *Issuer* fields available in the 5-tuple and 3-tuple are managed as plain text. Thus the authorization model is decoupled from the authentication method used to authenticate both *Users* and *Issuers*. They could be authenticated by any authentication mechanism such as OpenID, X.509 or Kerberos, among others. In our implementation we have used both X.509 and OpenID.

Object hierarchies [16] provide powerful expressiveness in the authorization model. It enables the extension of the scope of a privilege applied to a given object to all the objects who are descendants of the original object. The use of hierarchical paths is also a common programming abstraction used in cloud computing, for example REST [6]. To support this, the *Object* field allows the expression of the object hierarchy by means of *Paths*, using a special "\" symbol. A path defines the hierarchical structure of the protected resource. For example "*\pictures\imageA*" denotes that the *Pictures* object is a parent of the *imageA* object. In this case, *Pictures* is a folder and *imageA* is a file. However, these semantics are provided by the *Interface* defined in the 5-tuple. We use the "*" symbol to make it easier to define privileges applied to object hierarchies. So the "*" symbol in the path defines that the privilege is applied not only to the specified object but also to all its descendants in the hierarchy. For example, the (*Jose*, *user*(*Nigel*), *Read*, *CloudStorage*, *root*\∗) is interpreted as *Jose* says that user *Nigel* can *Read* the folder *root* and all its subfolders in the *CloudStorage* service. Then, the 5-tuple can now be defined as (*Issuer*, [*User*|*Role*], *Privilege*, *Interface*, *ObjectPath*).

To see how the above supports federation, consider a cloud computing scenario with a multi-tenancy authorization system, as depicted in figure 1. This scenario is composed of three different businesses. Each business has two different cloud services. Cloud services belonging to distinct businesses may use a different information model with information related to users, privileges, roles and resources. However, collaboration agreements between businesses (federation) could require the sharing of authorization information.

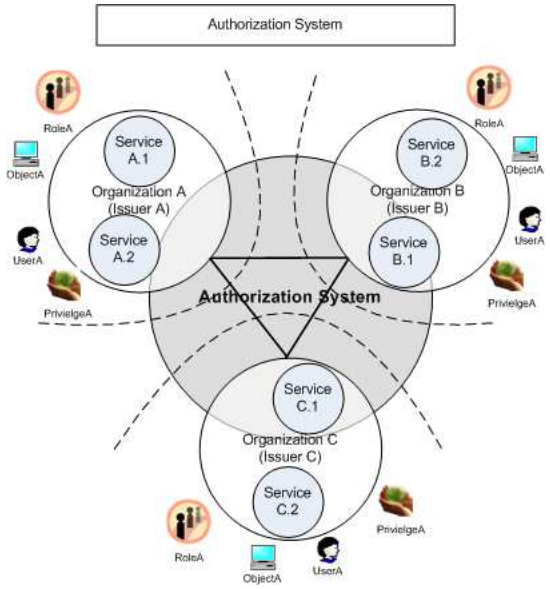

Figure 1. Example of Authorization System in a Multi-tenancy scenario

For example, the 5-tuple (*IssuerA*, *role*(*IssuerB*, *users*), *Read*, *ServiceA*.1, *\root*) can be interpreted as: *IssuerA* grants anybody with *role*(*IssuerB*, *users*) *Read* access to the *\root* folder of the file system provided by the service *ServiceA*.1. Note *role*(*IssuerB*, *users*) is controlled by Issuer B. Similarly we may have another 5-tuple: (*IssuerA*, *role*(*IssuerC*, *users*), *Read*, *ServiceA*.1, *\root*), for "C's users". This example entails many privacy concerns since it requires that *ServiceA*.1 can access the authorization statements of both *IssuerB* and *IssuerC*. This privacy is controlled by the *Trust Manager* module in the architecture described in IV.

All the fields with exception of *Issuer* field can be set up with "*" symbol to refer to "All" the instances available in the information model. For example, the following 3-tuple (*Jose*, *user*(∗), *Public*) can be interpreted as *Jose* says that all the users belong to *role*(*Jose*, *Public*).

The above model is intended to support multi-tenancy, role-base access control, hierarchical RBAC, path-based

object hierarchies and federation. The different issuers use this authorization model to define the authorization information in the system. When there is an authorization request, the authorization system uses all authorization information to prove if a request is authorized. If it cannot prove authorization using the stored 5-tuples and 3-tuples then a "deny by default" is applied rejecting the access to the resource. The next section describes our implementation architecture for this model.

IV. Architecture of the Authorization System

This section describes the implementation architecture of the authorization model described in section III. Figure 2 depicts the architecture. The system is composed of two layers.

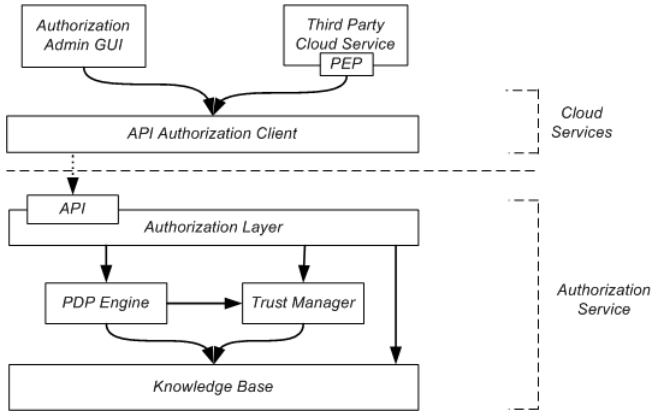

Figure 2. Architecture of the Authorization System

The first layer is the authorization server which provides authorization services to cloud services by means of the second layer. The second layer is the client API used for remote access to the authorization server. This API is used by the different cloud services which want to protect their resources using the authorization system. The intent is for the cloud services to implement their Policy Enforcing Point (PEP) using this API. The PEP component is a module of the cloud service in charge of the enforcing in the final system the authorization decisions. The same API is uses by the Authorization Administration GUI - it does not require any special or additional operations.

The authorization service is composed of several components. The *Knowledge Base* is a central repository which stores all the authorization statements and the role memberships, defined by the 5-tuples and the 3-tuples, respectively. Moreover, the *Knowledge Base* contains all the trust relationships - see section IV-A.

The *PDP engine* component is the core of the authorization system: it makes authorization decisions using the authorization statements and role memberships expressed in the authorization model described in the section III. To this end, the *PDP engine* retreives the authorization information from both the *Knowledge Base* and the *Trust Manager* components. The *Knowledge Base* provides all the authorization statements (3 and 5-tuples) whereas the *Trust Manager* determines which entities' authorization statements are used in the authorization decision.

Finally, an *authorization API* provides all the services related to the management of the trust, authorization grants, role memberships and authorization services. This *authorization API* is accessed by the *client API*. Secure interaction with the authorization system can be ensured by use of digital signature and encryption methods such as W3C XML-Digital Signature, W3C XML-Encryption and W3C XAdES or by means of end-to-end encryption protocols like SSL and TLS. All users, both those issuing authorizations and those requesting authorization decisions, are authenticated to control access to the authorization system. This protects the authorization system against unauthorized issuers as well as those requesting for authorization decisions.

A. Trust Model

A trust relationship, denoted as $A \precsim B$ (A trusts B), represents a collaboration agreement for which the issuer B can access authorization statements by A. This allows B to make use of statements by A in the knowledge base. So "A trusts B" here means "A trusts B to use its authorization statements" - it is used to control access to authorization statements to protect privacy. For example, in authorization decisions for B, the PDP will use the statement $role(A, users)$ if the trust relationship $A \precsim B$ exists, if the relationship does not exist, the statement will not be used by the PDP even if it exists in the Knowledge Base. This allows entities to control access to their authorization statements. They are only usable by entities which are explicitly trusted. The negotiation protocols among the parties to establish this trust, are carried out by external methods which are out the scope of this paper. By default no-one trusts anyone else unless there is an explicit statement of this in the Knowledge Base. Moreover, only an issuer A can insert or remove that $A \precsim someone$. Thus *Trust Manager* controls the privacy of all information stored in the Knowledge Base and when it can be used in the PDP component. **This basic trust model fosters simplicity and scalability. For this reason, this proposal does not cover intentionally a fine-grain control on trust between peers.**

B. Authorization Process

To understand the authorization process, consider an authorization request in which a user belonging to entity A is trying to access to a resource belonging to entity B. Moreover, entity A trusts entity B and vice versa ($A \precsim B$,$B \precsim A$). The sequence diagram for carrying out the authorization request is depicted in figure 3. In this example we assume that both A and B are using a common, multi-tenanted authorization service. The sequence diagram shows the intereaction of entity B's service in response to an invocation from entity A.

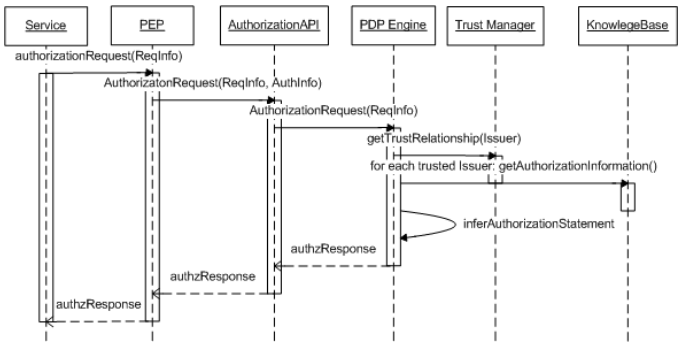

Figure 3. Sequence Diagram of an Authorization Request

Firstly, the user belonging to the entity *A* tries to access to a protected resource controlled by the PEP of a cloud service belonging to entity *B*. The PEP of *B*'s service establishes a secure communication with the authorization system using the *client API*. The API redirects the authorization request to the PDP component. The PDP component retrieves the trust relationships for the entity *B* (that is the list of entities which allow *B* to use their authorization statements for authorization decisions) from the *Trust Manager*. After that, the PDP uses this information to ask for the authorization statements to be used in the authorization process from the *Knowledge Base*. Thus, the *PDP* only retrieves authorization statements that B is allowed to see. Then the PDP applies the authorization model described in section III to make the authorization decision. Finally, the response is sent to the PEP of the entity *B* which is in charge of granting or denying access to the resource.

V. Implementation details

The implementation is composed of three different Java-based components, corresponding to those shown in figure 2: *Authorization Client API*, *Authorization System* and *Authorization Admin GUI*. The *Authorization System* uses a relational database to provide an optimized and scalable solution to manage efficiently a large *Knowledge Base* with authorization information. All access to the database uses the JDBC driver which makes it easy to change the underlying database. Currently, the HSQLDB[1] database version 2.0 is used as *Knowledge Base*. The tables used to store the authorization information have been highly indexed to improve system performance. The PDP engine makes the authorization decision has been implemented using SQL queries. These queries search authorization information in the *Knowledge Base*. For each authorization request the PDP constructs one or more queries to make the decision according to the authorization model explained in section III. The PDP engine uses the trust relationships provided by the *Trust Manager* component to build the correct the SQL query: queries are constructed so that only authorization statements from issuers who trust the entity requesting authorization are used. If an issuer has not issued a trust statement for the requesting party, its authorization statements will not be used. The *Trust Manager* keeps up-to-date the trust relationships between the different issuers into the *Knowledge Base* and retreives trust relationships using SQL queries against the *Knowledge Base*.

The authorization API is published by means of RESTful [6] web service technology which enables the remote invocation of the methods for the different parties involved. The list of the methods provided in this API provide methods to insert, remove and search information in the *Knowledge Base* including grants (5-tuple) and roles (3-tuple). Moreover, there are methods to manage the *Trust Manager* component: insert, remove and search trust relationships.

These methods determine authorization: *hasAuth*(), *hasGroup*(), *proveGrant*() and *proveGroup*(). The *hasAuth*() and *hasGroup*() methods provide a Boolean value representing whether or not the user is authorized to access to a given resource or has a particular role; the *proveGrant*() and *proveGroup*() methods also provide the explanation (inference trace) for why the users have been authorized: the list of authorization statements that provided the authorization. Notice the different between *hasAuth*() and *searchGrant*() methods. The former performs the inference process explained in the authorization model. For example the transitiveness between the grants associated to *Roles* and the *User* belonging to these roles, etcetera. The latter only search grants in the *Knowledge Base* without any inference process. This is useful for browsing and managing the Knowledge Base.

The *Authorization Client API* establishes a secure channel to the server using SSL. Authentication of client and server is via X.509 certificates. Note that the methods exposed in this API do not have an "issuer" parameter in any method. This is because the issuer is automatically inserted by using the X.509 name in the client's certificate. So clients cannot issue authorization statements on behalf of other issuers or identities - only identities that they can authenticate.

We have also implemented a graphical web user interface to enable end-users to use the authorization services in a friendly and intuitive way. They can use this to browse the knowledge base, as well as issue authorization statements and request authorization proofs.

VI. Performance Results

This section provides some performance results gathered from the prototype described in section V. The experimental set up is composed of two different machines: client and server. The server is an Intel Core 2 DuoT7500 2.20 Ghz 4Gb RAM with WinXP SP3 running the authorization server and the HSQLDB database server. The client is an AMD Opteron 252 2.59Ghz, 4Gb with WinXP SP3 running an application which emulates cloud services using the authorization server by means

[1]HSQLDB database is available at *http://hsqldb.org/*

Execution	1	2	3	4	5	6	7	8
I (Authorization statements)	10	100	1000	10000	2500	2500	2500	2500
A (authorization requests)	100	100	100	100	1000	1000	1000	1000
S (grant searches)	100	100	100	100	1000	1000	1000	1000
X (tasks)	210	300	1200	10200	4500	4500	4500	4500
T (threads)	10	10	10	10	1	10	100	1000

Table I
INFORMATION ABOUT THE DIFFERENT EMULATIONS EXECUTED

of the *client API*. Both systems are interconnected by means of a cross-over cable with a 1Gbit Ethernet link.

The emulator starts T threads. Each thread represents a cloud service using intensively the authorization system. All the threads are started in parallel in order to stress the authorization system with concurrent invocations. Each thread may execute a number of sequential tasks (X) representing the work load of each concurrent party.

The number of tasks executed in each thread is determined by parameters. Concretely, the emulator receives the following parameters: i) I number of authorization statements to be inserted; ii) A number of authorization requests to be processed; iii) S number of grant to be searched. The number of tasks to be emulated are $X = I + A + S$.

The emulation tries to recreate a realistic scenario. For this reason, I, the number of statements inserted in the authorization system uses the following distribution: 51% users, 11% roles, 60% paths, 2% interfaces. For example, in case $I = 100$, these 100 authorization statements contains, 51 different users, 11 different roles and so on. 20% of the authorization statements contain roles, rather than users as the subject, so in the case $I = 100$, 20% of the authorization statements will include one of the 11 different roles in the subject. **These percentages have been generated by observing our business environment to represent a realistic scenario**. The second and the third steps in the emulation perform the execution of queries. The second step invokes the *hasAuth*() method whereas the third step invokes the *searchGrants*() method. The information passed as parameters of the query methods is randomly selected. However, only 5% of the queries use roles in the *Subject* field because we expect it to be more common to ask about the authorization over a specific user (individual) as the subject than over roles as the subject. Table I shows the details of the different emulations.

Table I shows two sets of emulations: columns 1-4 and columns 5-8. The first set of emulations is intended to demonstrate how the authorization system scales with increasing numbers of authorization statements (up to 10,000). The second set of emulations is intended to demonstrate how the authorization system scales with increasing numbers of simultaneous users (up to 1,000).

The times measured include: i) creation of the REST packet in the client, ii) submission of the packet through the network, iii) reception and processing of the required action in the server, iv) submission of the answer and v) reception and parsing of the answer.

Figure 4.a) shows average invocation time for *hasAuth*

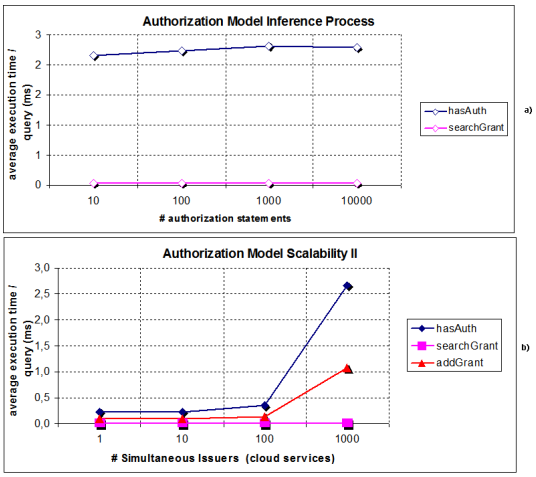

Figure 4. Scalability of the Authorization Model

and *searchGrants* for the first set of emulations (columns 1-4 of the table I). This scenario consists of ten simultaneous cloud service PEPs invoking the authorization system in parallel. There is an exponential increase in the number of authorization statements in the system. The intention is to analyse how the size of the Knowledge Base affects in the performance of the system. The results show that the execution time of searches and authorization requests is nearly constant even when there is an exponential growth of the Knowledge Base. This demonstrates high scalability of the system: in the case of the biggest Knowledge Base with 10,000 authorization statements it is able to serve 440 authorization requests per second.

Figure 4.b) shows the average invocation time for *hasAuth*, *searchGrants* and *addGrant* in the second set of emulations (columns 5-8 of the table I). This scenario consists of medium size authorization scenario composed of 4500 tasks (2500 authorization statements in the Knowledge Base, 1000 authorization requests and 1000 grant searches). There is an exponential increase of the number of simultaneous cloud service PEPs invoking the authorization service. The intention is to analyse how the response of the system varies with the number of simultaneous PEPs invoking the authorization services. The results demonstrate the scalability of the system. For example, the step from 100 to 1000 users (10 times more) lead to an increasing in the invocation time from 0.34 to 2.67 ms (7.8 times) in the worse case (*hasAuth*). In the latter worst case, the system still supports 370 *hasAuth* requests per second with 1000 PEPs simultaneously accessing the authorization system. The key to the performance of this architecture is the use of a database as the Knowledge Base with highly optimized indices.

VII. Conclusions and Future Work

The authorization model proposed in this paper supports multi-tenancy, role-base access control, hierarchical RBAC, path-based object hierarchies and federation.

These features are intended to provide a convenient authorization service for cloud services, especially those using path based paradigms such as REST [6], as the abstraction of the authorization model better matches the abstractions used to implement these cloud services. We have described the architecture and implementation of the model and demonstrated its scalability. We believe additional improvements in scalability could be made by experimenting with different underlying databases. The authorization system deals with trust and privacy issues that arise because of federation: authorization statements are private unless trust is stated explicitly.

Acknowledgement

Thank to Ministerio de Educacion y Ciencia and Fundacion Seneca for sponsoring the research activities under the FPU grant AP2006-4150, the research project TIN2008-06441-C02-02 and the funding program 04552/GERM/06.

References

[1] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[2] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? an architectural map of the cloud landscape," in *Proceeding at ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009.

[3] R. Chow, P. Golle, M. Jakobsson, R. Masuoka, J. Molina, E. Shi, and J. Staddon, "Controlling data in the cloud: Outsourcing computation without outsourcing control," in *Proceedings at ACM Cloud Computing Security Workshop (CCSW)*, 2009.

[4] J. D. Amit Sangroya, Saurabh Kumar and V. Varma, "Towards analyzing data security risks in cloud computing environments," in *Proceeding at International Conference on Information Systems, Technology, and Management (ICISTM 2010)*, 2010, p. 255265.

[5] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings at 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009.

[6] R. T. Fielding, "Architectural syles and the design of network-based software architectures," Ph.D. dissertation, University of California., 2000.

[7] S. Berger, R. Caceres, K. Goldman, D. Pendarakis, R. Perez, J. R. Rao, E. Rom, R. Sailer, W. Schildhauer, D. Srinivasan, S. Tal, and E. V. and, "Security for the cloud infrastructure: Trusted virtual data center implementation," *IBM Journal. Resources & Developments*, vol. 53, no. 4, p. 12, 2009.

[8] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, "Securing elastic applications on mobile devices for cloud computing," in *Proceeding at ACM Cloud Computing Security Workshop (CCSW)*, 2009.

[9] C. Danwei, H. Xiuli, and R. Xunyi, "Access control of cloud service based on ucon," *LNCS Cloud Computing*, vol. 5931, pp. 559–564, 2009.

[10] R. Sandhu and J. Park, "Usage control: A vision for next generation access control," *LNCS Mathematical Methods, Models, and Architectures for Network Security Systems*, vol. 2776, p. 1731, 2003.

[11] L. Hu, S. Ying, X. Jia, and K. Zhao, "Towards an approach of semantic access control for cloud computing," *LNCS Cloud Computing*, vol. 5931, p. 145156, 2009.

[12] S. D. Capitani, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of access control evolution on outsourced data," in *Proceeding at International Conference on Very Large Databases*, 2007.

[13] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *29th IEEE International Conference on Computer Communications*, 2010.

[14] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The nist model for role-based access control: Towards a unified standard," in *Proceedings of the fifth ACM workshop on Role-based Access Control*, 2000.

[15] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224–274, 2001.

[16] Q. Ni, E. Bertino, J. Lobo, and S. B. Calo, "Privacy aware role based access control," *IEEE Security and Privacy*, vol. 7, no. 4, pp. 35–43, 2009.