Identity Access Management for Multi-tier Cloud Infrastructures

Mohammad Faraji, Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada
Email: {ms.faraji, joonmyung.kang, hadi.bannazadeh, alberto.leongarcia}@utoronto.ca

***Abstract*—This paper presents a novel architecture to manage identity and access (IAM) in a Multi-tier cloud infrastructure, in which most services are supported by massive-scale data centres over the Internet. Multi-tier cloud infrastructure uses tier-based model from Software Engineering to provide resources in different tires. In this paper we focus on design and implementation of a centralized identity and access management system for the multi-tier cloud infrastructure. First, we discuss identity and access management requirements in such an environment and propose our solution to address these requirements. Next, we discuss approaches to improve performance of the IAM system and make it scalable to billions of users. Finally, we present experimental results based on the current deployment in the SAVI Testbed. We show that our IAM system outperforms the previously proposed IAM systems for cloud infrastructure by factor 9 in throughput when the number of users is small, it handle about 50 times more requests in peak usage. Because our architecture is a combination of Green-thread and load balanced process, it uses less systems resources, and easily scales up to address high number of requests.**

I. Introduction

Cloud computing has promoted the hosting and delivery of services over the Internet and the movement of computation and data from terminal devices and local servers to core data centres due to advantages in flexibility, scalability, and economic of savings [1]. Most services have been supported by massive-scale distant datacenters located at sites. However, some services will require low latency (e.g. alarms in smart grids, safety applications in transportation, monitoring in remote health, fire or emergency alarms in smart cities), the processing of large volumes of local information (e.g. video capturing in lecture rooms), or high security provided by intelligent converged network and computing at the edge of the network, for example in the premises of traditional telecom service providers.

The Smart Applications on Virtual Infrastructure (SAVI) project has been established with a focus on future application platforms designed for applications enablement [2]. As shown in Figure 1, SAVI considers a multi-tier cloud infrastructure to include Smart Edges where local players work together with remote massive-scale data centres to provide better Quality of Service (QoS) for sensitive applications. SAVI investigates the hypothesis that all computing and networking resources can be virtualized and managed using Infrastructure-as-a-Service (IaaS). The Smart Edge should go beyond conventional cloud resources to address QoS demanding applications such as video distribution, fast and secure communication, wireless access controls, etc. In other words, the Smart Edge will be heterogeneous data centres including line rate processors, re-configurable hardware, graphical processors, specialized hardware accelerators and crucially future highly programmable networking equipment.

978-1-4799-0913-1/14/$31.00 © 2014 IEEE

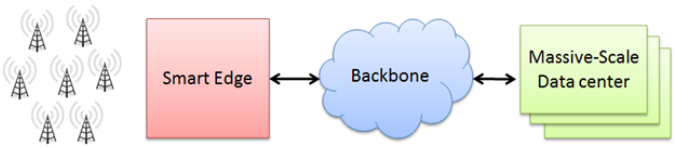


Fig. 1. Multi-tier cloud infrastructure

The Global Environment for Network Innovations (GENI) [3], which provides a virtual laboratory for networking and distributed systems research and education, is another multi-tier cloud and networking infrastructure but with a different view. Unlike SAVI which is looking for a unified management layer, GENI intends to federate a variety of testbeds or resource providers with different control and management planes. GENI provides a wrapper in front of a set of testbed suits or resource aggregators to provide unified APIs for users, while maintaining their independence. Users can allocate slices in different testbeds by joining to one of the organizations in GENI. GENI can be considered as a community cloud.

While the benefits of cloud computing is clear, security is a severe concern in these infrastructures. Kandukari et al.[4] considers five cloud security issues that should be included in a Service Level Agreement. There are the following: privileged user access, data location, data segregation, data disposal and investigation and protective monitoring. Privileged user access ensures only authorized users have access to an organization data and resources. Therefore, identity and access management is considered as a security concern in cloud computing. Various models have been proposed to address identity management in clouds, such as central IAM, trusted third party, federation solutions, etc. Most of solutions are mainly focused on federation of cloud providers, and pay less or no attention to access management.

In this paper, we propose a new architecture to manage identity and control access to resources in a multi-tier cloud infrastructure. First, we discuss system requirements, and then we propose an architecture to address these requirements. Our architecture comprises two major components: middleware and central IAM to manage user and infrastructure related data. Middleware sits in front of a resource provider and handles time-consuming decision making such as authentication and

authorization, while the repository handles data manipulation. We deploy performance enhancing techniques to boost middleware and repository performance while using load balancing to make it scalable. This architecture has been implemented on SAVI testbed that spread throughout Canada.

In summary, this paper makes the following contributions:

- We design a comprehensive architecture for a centralized IAM in a multi-tier cloud infrastructure.
- We propose three performance enhancing techniques of the IAM for scalability: load balancing, caching, and middleware.
- We propose a fine-grained policy-based access control for trust-relationship management.
- To test our proposed IAM on the Canadian SAVI Testbed.
- Using the testbed, we have shown that the IAM outperforms the previously proposed IAM for cloud infrastructure in throughput and response time.

This paper is organized as follows. Section 2 reviews various cloud- and testbed-provider IAM systems. Section 3 describes IAM requirements and our design. Section 4 focuses on scalability issues and our solution approaches. Section 5 shows performance evaluation results for validation. Finally, we present conclusions and possible future work in Section 6.

II. Related Work

Identity management (IDM) in web services is experiencing a paradigm shift from organization-centric to trusted third party on the cloud. Third party identity providers allow both scalability, and flexibility to users and applications over the Internet. Traditional identity management models are either user-centric or service-centric. In user-centric models, a user manages multiple identities through software or a hardware pluggable authentication device [5]. In service-centric models, this job is offloaded to service providers which can either store the credentials centrally or can be federated. These two approaches are used in the cloud as well. Angin et al. [6] propose an entity-centric approach for IDM in the cloud which leverages active bundles to send sensitive data and policy to a service provider.

There are several solutions for IDM in clouds. Gunjan et al. [7] compiled a list of available technologies and solutions for cloud computing, including Primary and Identity Management for Europe (PRIME), Windows CardSpace, OpenID, Higgins, and Liberty Alliance. Academic research has concentrated on cloud-based IAM security issues. Huang et al.[8] tries to avoid sending clear text to other cloud providers by designing a PPID protocol where IDs are divided between services, enabling a cloud provider to retrieve user information while preserving a users privacy. Other research works focus on the general concepts of IDM and federation: Huang et al [9] proposed an identity federation broker to allow federation of in-cloud services, external services, and on-premise resources. In [10], the authors are building a distributed identity management model for a collection of institutions to collaborate.

ExoGENI is a GENI testbed to build a network of cloud providers based on OpenStack software. Network campuses can be served through ORCA [11] control framework by connecting to network circuit fabrics. EXoGENI IAM uses libabac to add attribute-based access control to the testbed, and authorizes entities according to identity, affiliation, and type of the activity. It also supports delegation in the format of capability-based access control. EXoGENI uses a PKI infrastructure to manage identities and authenticate entities. ExoGENI IAM leverages Shibboleth to to be federated with other infrastructure and cloud provider [12].

Current approaches to cloud IAM are concentrating on offering solutions on particular issues such as federation or finer-grained access control. The lack of a comprehensive analysis, from conception to physical implementation, to incorporate these solutions resulted in impractical and fractured solutions. Meanwhile, some elements of the identity and access management problem are specific to advanced cloud platforms such as SAVI. For example, infrastructure users can acquire a set of virtual resources to run their applications, or an application can send requests to resources on behalf of users. A virtual resource can span multiple providers in different geographical locations or under another administration. In this paper we focus on designing and implementing a comprehensive architecture that meets multi-tier cloud requirements such as SAVI.

III. Identity and Access Management

We present an architecture that improves the security in a multi-tier cloud infrastructure where resources are typically engaged by applications on the behalf of users. For this reason we concentrate on application requirements and we consider the requirements of a user-centric IDM. The SAVI IAM adopts IdP/SP model where a central Identity provider provides information for a service provider. In a multi-tier cloud orchestration, the resources are the service providers and the IdP is the SAVI IAM system. The main challenge in IdP/SP model is scalability and our solution is to extend the model to incorporate middleware that assume some of the IAM responsibilities.

A. Requirements and Objectives

There are three categories of requirements for a cloud-based IAM: 1) authentication and federation, 2) access management, and 3) manifesting and reporting. When an application runs in the cloud, it initially identifies itself to the IAM to receive a list of available resource. Next, the application sends its request to acquire a slice of the selected resource. The behaviour of the applications is audited for future reference or for rolling back the system to a stable state in case of system crashes.

Authentication is the process of verifying an entity identity to ensure that the entity has enough permissions to access the resource. Authentication is also the primary issue in federating with other cloud providers. The IAM needs to support a standard way of authenticating entities, such as the Security Assertion Markup Language 2 (SAML2), in order to provide a cross-domain service. However, for the internal communication, applications may use different authentication

mechanisms to identify themselves to the IAM. In order for the IAM to support any authentication mechanism, it is necessary to separate the authentication logic from the credential format. Strong Authentication is a major security concern for a an IAM, and so the SAVI IAM must have the ability accept a combination of various authentication methods.

The second category of requirements deals with *access management*. A goal of cloud infrastructures is to provide a highly-programmable platform, so the use of a simple coarse-grained access control would be overly constraining to users and applications, and refrains administrators to implement the least-privilege and separation-of-duties principle. Furthermore, since the rate of resource allocation and deallocation can be very high in an application-oriented infrastructure, the authorization layer should not inflict a considerable performance overhead on resource providers. The authorization layer must incorporate the least-privilege and separation of duties design principles.

The last category is *manifesting and reporting*. IAM is considered as the point of entry for a user, and it should provide sufficient information about the infrastructure to allow the user to proceed, such as endpoints to reach resources, their geographical locations etc. On the other hand, since IAM middleware is intercepting all requests, it can track and furnish data for all basic audit requirements, as well as to identify access violations or attacks, and to quantify risks and threats.

B. Proposed System Design

We have developed an IAM solution for the multi-tier infrastructure such as SAVI testbed. SAVI Testbed main entities includes a SAVI TB Control Centre, Core Nodes, Edge Nodes, and a SAVI network. Core and Edge nodes contain resources that are used for creating applications. The SAVI TB Control Centre hosts SAVI control and management functions including resource allocation, clearing house, monitoring and measurement, and so on. The SAVI IAM system located in the SAVI Control Centre, and asserts identities about users, applications, and a threads of execution that can be called an entity.

We have designed and implemented SAVI IAM to address the requirements in the previous section. The SAVI IAM is a central identity manager with distributed middleware based on IdP/SP model and comprised of 6 basic components: Manifesting Management, Identity Management, Policy Management, Token Management, Authentication Management, and Middleware. Figure 2 illustrates how these components interacts with each other.

To access a resource, an entity needs the resource URL-named endpoint. These endpoints, together with associated resource attributes such as geographical locations, quota, service level agreement, service type, etc., are enclosed in a data structure called an endpoint record. Each endpoint record has three types of endpoints: Public, Admin, and Internal: Public for external access to the resource, admin for administrative tasks bounded to a management network, and internal for inner components invocations. The list of the endpoint records is formatted and placed in Service Catalog which enable a user to choose a resource aligned with the governance, the standards and the requirements. The list can be updated, retrieved, and created through the manifesting management component. The service catalog is divided into regions that represent edge nodes in SAVI. Each edge node is independent of other nodes in terms of provisioned resources.

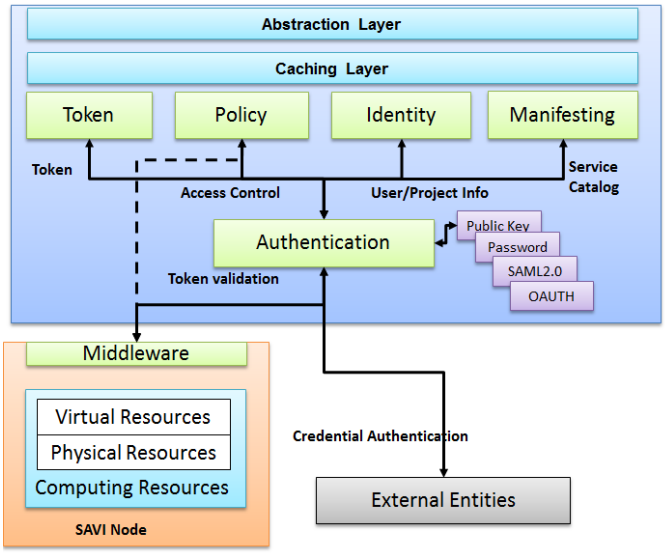


Fig. 2. IAM Architecture

The second component is Identity Management which is used to store, and manipulate users, projects, and associated attributes such as role. Likewise, the policy management component stores access management policies, and through it, users can perform CRUD (Create, Retrieve, Update, Delete) operations on a policy. During the authentication process, the SAVI IAM returns a security token as part of the security context. The Token Management generates a token for the authentication service. It supports the following token format: Universally Unique Identifier (UUID) and Public Key Infrastructure (PKI). The UUID is a series of letters and numbers to maintain session, and the middleware needs IAM to validate the UUID token. But, the IAM PKI generates a certificate for the entity and signs it with its Public Key, therefore, the middleware does not invoke IAM to validate the token. The token manager also provides APIs to retrieve token-associated data such as a username, project etc, which is used by other APIs to obtain security context.

Token Management in concert with the authentication module are the cornerstone of the authentication process. As shown in Figure 2, the authentication module supports multiple authentication mechanisms. It fetches a users attributes and project information from the identity module to validate a users credential. Separating logic from storage enables IAM to support different authentication methods without any mandates on the credential storage layer, and helps to create a flexible authentication module. The IAM authentication module determines the method of authentication in runtime. In each request, there is an identity-type field that defines the method to validate the credential. Users can choose different methods of authentication (e.g. Public Key, Password, OAUTH, and SAML2.0), and correspondingly, the token format of can be different from a request to another request.

Middleware resides in front of a resource provider to cap-

ture requests. After a request identity and access privilege are confirmed, the request is handed over to the service provider; otherwise, the it is directly rejected. As shown in Figure 3, our IAM has two middlewares: authentication and authorization. Authentication middleware validates the security token, and authorization checks the requested action against the policy. To improve performance, we are maintaining authentication in the IAM while we delegate authorization to a resource provider and transfer the policy to it.

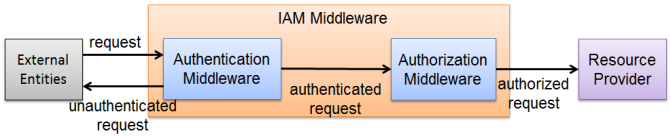


Fig. 3. IAM Middleware Architecture

C. SAVI Access Control

The access control model determines who can access to what resources on the cloud infrastructure. Access control is a collection of policies and components used to implement the access control model in the cloud. SAVI enables administrators to define access control policies and attributes in a completely isolated environment to provide flexible, adaptable, and fine-grained access to their users and applications. Isolation provides a way to address the problem of containment for compromised applications [13] and has been used before to have different permissions for the same role in Role-based access control [14]. Isolation allows administrators to define authorization parameters (e.g. role) in an isolation scope. The isolation scope is the granularity of isolation and can be a project, an administrative domain, or a physical node. As shown in Figure 3, When the user is authenticated, along with his credentials, the authorization middleware task his attributes to create the security context. It also synchronizes the local copy of policies with the SAVI IAM in the context of an isolation scope.

As illustrated in Figure 4, the authorization middleware comprises four essential components: Policy Enforcement Point (PEP), Policy Decision Point (PDP), Attribute Assertion, and Policy Management Point in the IAM. When the PEP captures a request, it forwards the request to PDP to perform authorization. The PDP retrieves the security context from the authentication middleware and resource attributes from the Attribute Assertion module. It stores a local copy of the access control policy, and updates it periodically with IAM over Policy management component.

Bear technology-independence in mind, PDP is able to use different access control models to authorize the request. We have implemented two access control models in SAVI so far: Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC).

1) Role-Based Access Control: Role-based Access Control(RBAC) [15] is very useful model for cloud computing. Although its benefits is fundamentally limited to entity attributes, it simplifies the management of permissions. The main concept of RBAC is the granting of permissions to users according to the associated roles. Therefore, roles is a layer

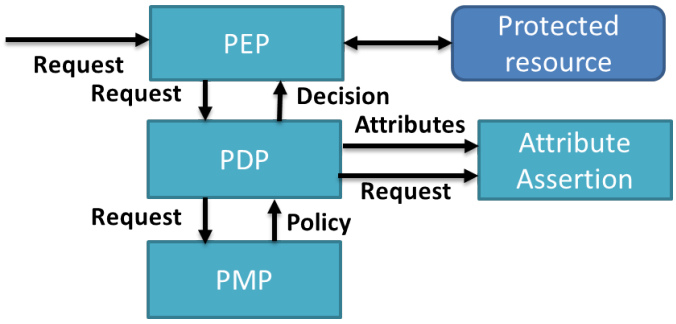


Fig. 4. Authorization workflow

of abstraction between users and permissions. Administrators assign roles to users, and roles incorporate new permissions as applications and systems join to them. There are three operations in RBAC: role assignment, request authorization, and role-permission association.

SAVI has implemented RBAC96 [16] which is Role-based access control with constraints. SAVI RBAC is implemented at API level. The required attributes to complete authorization in SAVI RBAC are the list of user roles, the user project, and the status of his identity. The authorization is done by checking the attributes against the policy which maintains the list of authorized roles with their domain names or any other isolation scope.

There are three drawbacks with RBAC: 1) it needs a substantial effort to define roles and assign them to users 2) an administrator needs to introduce a new role to cover discrepancies, leading to role explosion problem[17] 3) RBAC is not suitable for a cross-domain access management because reaching to agreement that what permissions are associated with a role is difficult [18]. Due to these barriers and to provide a finer level of access control, many service-oriented platforms have shifted to Attribute Based Access control.

2) Attribute-Based Access Control: To provide fine-grained access control, the logic needs to be more discriminating, and finer-grained in terms of what should be considered for a request to move forward. Instead of considering just one attribute (Role), several attributes can be taken into account (e.g. project, risk level, resource attributes). Attribute Based Access control (ABAC) [19] takes three categories of attributes into account: subject attributes (e.g. role, project), resource attributes (e.g. owner of a file), and environmental attributes (e.g. risk level). The relationship of these attributes is expressed in a predefined grammar in policy. To be backward-compatible, we have used the same notations as in RBAC.

To keep the attribute definition flexible, the IAM enables each component to defines its own attribute inside a data structure named resource map. The authorization middleware recognizes an attribute as long as it knows how to fetch and check it against the policy. A resource map is filled by Attribute Assertion module in which each resource defines how to provision an attribute. The attribute assertion module varies from a resource to another one, and is invoked by authorization middleware during access time. The authorization middleware calls a checker function to validate an attribute. A checker function directs the middleware how to interpret an attribute.

The middleware gives value of attribute along with the policy as an argument to the checker, and it returns a Boolean value to indicate approval or denial.

The middleware can apply policies in two ways: to protect a function or to filter the output of a function. When a policy wants to protect a function, it intercepts a request before calling the function, and if the request is authorized it goes through, which can be translated to a binary protection either accepts or denies it. While in filtering, policy is used to remove unauthorized output of a function. Parsing the policy is a time-consuming and complex job. To parse the policy, we use a greedy reduction algorithm to reduce a sequence of tokens into a single terminal, the value of which will be the root of the decision making tree. We divide a policy into tokens, and map each token to a check function.

D. Delegation and Trust

Delegation of duty allows to perform a task when the initially-assigned user is not available to complete the task. It does so by granting an entity (user or process) an ability to perform an action temporarily. Delegation comes handy when an administrator wants to authorize an application or user at setup time to perform some errands in the future, long after a token is expired. There are two types of delegation: Dynamic Delegation of Authority (DelAuth) and Dynamic Delegation of Action (DelAct). DelAuth grants all or part of a user authority (delegator) to another user (delegatee) to do something. DelAct grants a user the permission because the request is part of a process that is authorized. There are three types of situation where a delegation is required: backup of role, decentralization of authority, and collaboration of work. Failover application is a backup role to recover the system on the behalf of administrator when he is not available. Decentralization of authority helps to distribute functions in a group to implement the separation-of-duties principle, and collaboration of work occurs when two parties work on the same resource for a period of time.

The SAVI IAM supports Permission-based Delegation Mode (PBDM) [20] which is a model for delegation of authority in RBAC. It is based on RBAC96 and a user can delegate a privilege by delegating a role associated with the permission to one or more other users and confining that permission to a project. There are two types of role in SAVI: regular role and delegation role. When a user wants to delegate a role, it creates a delegation role and assigns it to another user. In SAVI, the unit of delegation is permission that corresponds to a delegation role. Users can also assigns their regular role that contains a set of permissions. By default, administrators can delegate roles and permission; however, members need explicit authorization to delegate their roles. Delegation roles are created by an administrator, therefore, an administrator can control the permission flow by mapping permissions to delegations roles.

IV. Scalability and Performance

A multi-tier cloud infrastructure needs to scale to billions of transactions for millions of users and applications identity. Therefore, a centralized identity manager may be the bottleneck of a multi-tier cloud computing because if it slows down, the interaction with whole process slows down. An IAM in cloud infrastructure should dynamically scale up and down, so no entity should hoard resources in peak hours. The features introduced in Section 3 slow down the handling requests, so we need to introduce performance enhancing techniques to optimize operations and scalability mechanisms to keep the performance at the same level during the peak usage.

To maintain the system performance, we distributed decision making between the central IAM and the middleware in the way that the central IAM carries out authentication, while authorization is in middleware on the resource side. To make a system scalable, we need to consider points of failure. A slow central IAM can inflict delays on operations; therefore, we must optimize IAM throughput in the first place, and avoid redundant calls to it at the second step. Token authentication is the most frequent operation, and it is followed closely by credential authentication. Policy is a fairly rare operation, and it is placed the bottom of the list. User-, tenant- and role-membership occur at the same rate but significantly lower than the authentication process. Finally authorization is performed by middleware on a resource, and therefore it is not a matter of discussion in the IAM.

The performance of resource provider is heavily dependent on the performance of the central IAM due to authentication. Therefore, two strategies should be followed to boost this performance:

- Optimizing response time of the IAM
- Decreasing the number of calls to the IAM

To optimize response time, we identified time consuming activities in the IAM in order to produce the response. The response is composed of two pieces of information: user security context and service catalog. In order to return a response: 1. Credential info should be retrieved; 2. Token and service catalog should be formatted. The service catalog is fairly huge chunk of data because it contains the lists all available resources and associated attributes on the cloud, and since it is depends on user info(e.g. Project id), it has to be formatted for each user separately. Reducing the number of querying and formatting service catalog can decrease response time too.

To decrease the number of calls to IAM we have to store some data in the middleware. An examination of the authentication middleware workflow shows that we do not need to authenticate each token for individual requests as this token is valid for a period of time. Furthermore, instead of validating one token in each call, we need to validate a bunch of tokens. Therefore, the middleware must be cache coherent which means it should perform the authentication locally while not violating the semantics of the operation. Along these strategies, we leverage caching, offline token validation to boost performance. Load balancing over Green-Threads is our scalability-enhancing practice that enable IAM to address large number of requests.

A. Caching Layer

Caching is a layer in IAM to prevent redundant access to the database, and to avoid unnecessary data formatting. When a request needs data from the IAM, the IAM queries the database

to fetch, after formatting the data, the IAM sends it back to the user in the response payload. Fetching and formatting are two processing-intensive jobs when they come to handling millions of requests and large amounts of data. The Service Catalog is the largest chunk of data that IAM returns to a user. The Service Catalog is different for users and projects, therefore we cannot cache data at the datastore, and formating the service catalog takes more time than fetching it. We cache the data transparently after formation. However, transparency can result in inconsistency in data because other functions modify this data as well.

Our layer caches output of a function that queries and formats the data from the datastore, caching layer puts the output of each function in a distinct namespace to keep track of cached data. It provides some handlers to refresh cached data to overcome inconsistency. Other components invoke these handler to indicate the caching layer that a namespace data is not valid anymore.

B. Middleware

To validate tokens from regular entities, the middleware should connect to IAM and get a privileged token to verify the users token. This job costs three operations for the middleware. As tokens are valid for specific duration, the middleware can cache a token until it is expired or revoked by the IAM. When a request comes in, the middleware tries to find the token in the local cache, if the token is found, the middleware lets the request go through; otherwise, it has to be validated by the IAM. After validation, the token is cached in the local token lists. The middleware periodically updates the list of revoked tokens to be consistent with the IAM. Local caching reduces the overhead of connection establishment,

The performance overhead of internal authentication is huge during peak usage because resources need to call other resource providers to perform their jobs in multi-tier cloud computing infrastructure. Computation needs to connect to a networking component to create a network, and as result, it inflicts a huge number of unnecessary authentication on the system, and slows the system down enormously. To overcome this issue, we have introduced the concept of trusted parties where agents or trusted components are introduced to middleware of a resource, and they get through without validation. A trusted party is identified by its source IP address and API keys where are given as configuration manager to the resource. We refrain from performing authentication and authorization on a trusted party request to increase the performance.

C. Off-line token Validation

When the IAM issues a PKI token, it signs the token with its Public key. Each PKI token has a expiration date to indicates until when it is valid. The middleware does not need the IAM to verify a PKI token. Therefore, it can validated without IAM as well. The PKI tokens are issued to components or application that need to use a resource for a certain period of time. As the IAM can not revoke the PKI tokens, it does not issue them for users.

D. Load balancing

The using of load balancer is twofold. We keep the performance at the same level when the number of requests increased. In case of crash, the load balancer can as a failover mechanism. As the IAM instances is running in a separate processes and memory space, they can share the status of requests with each other. As a result, entities may be constantly receiving Unauthorized error when a process cannot find the state of the token request. To deal with this issue, IAM places tokens and security data on memcache under the same namespace. Namespace is caching space that the caching layer dedicate to a piece of data. The caching layer keeps track of cached data in the namespace. A caching namespace has three elements: name, list of keys, and cached blobs. For example, the namespace name for caching service catalog is ”catalog”, and list of keys is stored in the caching layer to facilitate the cached blob revocation. A caching blob is composed of access key and cached value. Because the caching layer is shared between the independent processes, if the processes keeps the security information in the caching layer, different instances can access to the security information by leveraging namespace and encryption parameters used to secure cached data. A load balancer uniformly distributes requests among instances, and also creates more instances when the number of requests for each instance passes a certain thresholds. As a result, the IAM system is able to dynamically scale up and down by increasing the number of backend processes.

V. Performance Evaluation

In this section, we evaluate the performance of our IAM using the SAVI Testbed [21]. We first describe our experimental setup for the evaluation, and then present our experimental results.

A. Experimental Setup

SAVI IAM is built on top of Openstack Keystone V2.0 [22]. Keystone is Openstack identity manager, and is composed of three basic components: Catalog, Identity, and Token. Service catalog stores a list of endpoints, and identity backend verifies tokens and credentials and handle CRUD operations on users metadata, token backend generates tokens. Keystone v2.0 does not support policy CRUD operations and PKI at the moment. Keystone supports a well-defined format for credentials, and can generate tokens in OpenStack and EC2 format. Keystone has two kinds of APIs which serve different types of users: Admin API to serve administrative operations and Public API for regular users. It has an authentication middleware that sits on in the front line of each resource on the cloud, this middleware authenticate users tokens through Keystone.

To carry out load testing on SAVI IAM, we utilized Apache JMeter that is a Java application used to carry out performance analysis on a variety of applications including: HTTP(s), SOAP, JMS, JDBC, and LDAP etc. It performs load testing and robustness testing to measure the behavior of an application under different load types [23].

To run these SAVI IAM, we used a xeon machine with 12 processors and 32 GB memory on SAVI testbed. To evaluate the performance, we conducted distributed testing on IAM

because a single client machine is unable, performance-wise, to simulate enough users to stress IAM. Therefore a master machine can control multiple, remote JMeter engines from a single JMeter GUI client. Each engine is an xeon system with 4 processors 4 GB memory on SAVI. The master is a large virtual machine with 4 processors and 8 GB memory.

B. Experimental Evaluation

In this section, we present the performance evaluation results and we compare the SAVI IAM performance with the baseline OpenStack Keystone v2.0. The performance comparison is to evaluate the throughput and response time of the SAVI IAM over Openstack Keystone. In each test experiment, we measure the following parameters for an individual credential authentication:

- Throughput: the number of requests per second the IAM or middleware has processed
- Average Response Time: the total running time divided by the number of successful requests sent to the IAM or Middleware

The evaluations are done on three different systems: 1) The baseline system which is Keystone v2.0. 2) A single instance of SAVI IAM with Caching-Layer enabled 3) A Multi-instance full featured SAVI IAM (Caching-Layer enabled) .

For all experiments, we assumed 300 projects with 10 regions (SAVI nodes), while each region has 10 endpoints which can be translated to total of 100 endpoints. The system performance is measured while increasing the number of concurrent fast clients from one client to 256 clients. Fast clients are dumb processes that do not process response, and continuously send requests to the SAVI IAM.

Figure 5 shows the evaluation results of the baseline Keystone v2.0 system. As it can be seen, the baseline system performance degrades significantly as number of users increases. This can be observed by rapid degradation of throughput measured in requests served per second (right side y-axis) and high increase in response time measured in milliseconds (left-side x-axis). When the number of concurrent fast clients passes 128, the number of successful requests are decreased due to timeout in serving them.

Figure 6 shows the performance of one instance of SAVI IAM with Caching-Layer enabled. As described before, The Caching Layer performs transparent caching of the data to put in the response of a request.

As it can be seen, the throughput of one SAVI IAM instance with Caching-Layer enabled is higher than that of Keystone v2.0 when number of concurrent users are high. When the number of the requests are low, Keystone v2.0 slightly outperforms SAVI IAM since SAVI IAM places a version of response in cache as well; however, when the number of requests increases and most of requests are repeating, the caching mechanism boosts system throughput because there is no need to query the database and format the data again. For requests in the range up to 64, the throughput of the SAVI IAM system is 4 times higher than that of the Keystone v2.0. In terms of response time, with the increase of requests, response

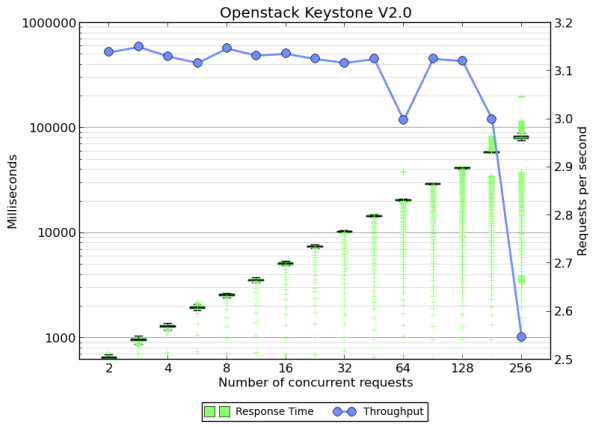


Fig. 5. Keystone v2.0

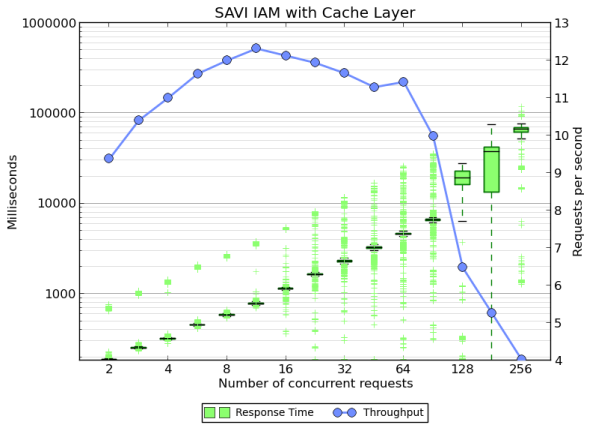


Fig. 6. Single Process IAM

time for the SAVI IAM is lower because fetching from cache is faster than querying the storage and formatting the data.

Figure 7 shows the performance of the full featured SAVI IAM with load balancer and 10 instances of SAVI IAM processes including the Caching Layer. As it can be seen, SAVI IAM significantly outperforms baseline in terms of both throughput and response time especially in high loads. Also a comparison of Figures 6 and 7 shows that the throughput is increased by a factor of 9 when 10 IAM instances are used. Moreover, the service time is decreased by a factor of 10 at the higher request rates.

Lastly, in Figure 8(a,b), we present the throughput and average response time for all three evaluated systems in one graph for better performance comparisons. This figure clearly shows that SAVI IAM is able to handle high request rates while keeping low response time and is able to scale up as needed to handle higher number of concurrent users.

VI. Conclusion and Future Work

In this paper, we have introduced a novel architecture for a cloud-based IAM. Our architecture uses a central IAM and decentralized middleware to carry out IAM duties. The main features in this architecture are scalability, adaptability in

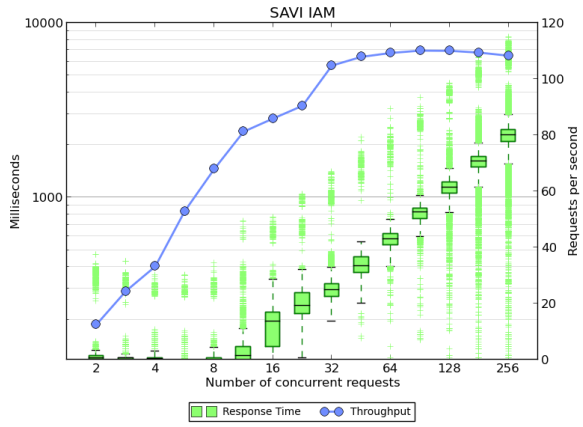
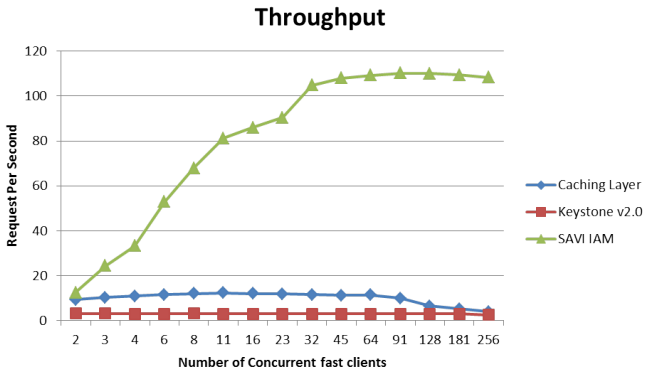
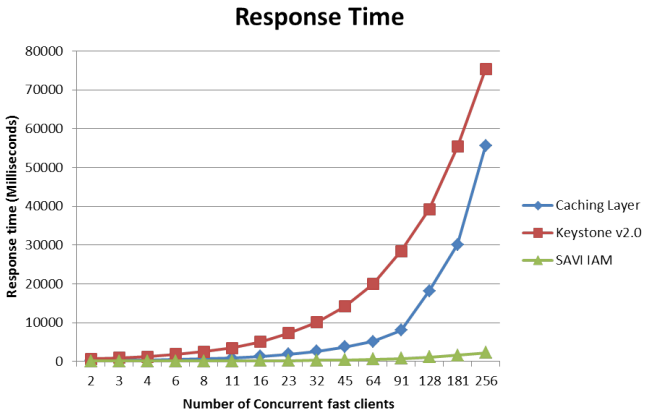


Fig. 7. Full-featured SAVI IAM



(a) Throughput



(b) Response Time

Fig. 8. Comparison of three evaluated systems

access control by implementing RBAC and ABAC, and extensibility to future technologies. The SAVI IAM introduces great flexibility in selecting authentication method by designing a common Pluggable Authentication module. It also introduces an open platform for authorization and delegation that is an essential requirement for an application-centric infrastructure. To make the IAM performant and scalable, we leveraged efficiency of Green-thread and scalability of load balancing. Our experiment shows SAVI IAM improves the throughput of Openstack Keystone by 10 times in off-peak usage, and 50 times in peak usage. In the future, we plan to study federation to enable bursting to other cloud providers. We are also interested in including new authentication technologies such as QR-Code and mobile device into SAVI IAM.

ACKNOWLEDGMENT

The work for all the published papers which follow is funded in part or completely by the Smart Applications on Virtual Infrastructure (SAVI) project funded under the National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks grant number NETGP394424-10.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[2] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Savi testbed: Control and management of converged virtual ict resources," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 664–667.

[3] T. Anderson and M. K. Reiter, "Geni: Global environment for network innovations distributed services working group," 2006.

[4] B. Kandukuri, V. Paturi, and A. Rakshit, "Cloud security issues," in *Services Computing, 2009. SCC '09. IEEE International Conference on*, 2009, pp. 517–520.

[5] A. Jøsang and S. Pope, "User centric identity management," in *AusCERT Asia Pacific Information Technology Security Conference*. Citeseer, 2005, p. 77.

[6] P. Angin, B. Bhargava, R. Ranchal, N. Singh, L. B. Othmane, L. Lilien, and M. Linderman, "An entity-centric approach for privacy and identity management in cloud computing."

[7] K. Gunjan, G. Sahoo, and R. Tiwari, "Identity management in cloud computing–a review," *International Journal of Engineering*, vol. 1, no. 4, 2012.

[8] X. Huang, T. Zhang, and Y. Hou, "Id management among clouds," in *Future Information Networks, 2009. ICFIN 2009. First International Conference on*. IEEE, 2009, pp. 237–241.

[9] H. Y. Huang, B. Wang, X. X. Liu, and J. M. Xu, "Identity federation broker for service cloud," in *Service Sciences (ICSS), 2010 International Conference on*, 2010, pp. 115–120.

[10] H. Koshutanski, M. Ion, and L. Telesca, "Distributed identity management model for digital ecosystems," in *Emerging Security Information, Systems, and Technologies, 2007. SecureWare 2007. The International Conference on*. IEEE, 2007, pp. 132–138.

[11] J. Chase, L. Grit, D. Irwin, V. Marupadi, P. Shivam, and A. Yumerefendi, "Beyond virtual data centers: Toward an open resource control architecture," in *in Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library), ACM*, 2007.

[12] T. F. Steve Schwab, "Managing identity and authorization for community clouds," Duke University, Tech. Rep., 2012. [Online]. Available: www.exogeni.net/

[13] P. Liu, S. Jajodia, and C. D. McCollum, "Intrusion confinement by isolation in information systems," *Journal of Computer Security*, vol. 8, 2000.

[14] N. Gunti, W. Sun, and M. Niamat, "I-rbac: Isolation enabled role-based access control," in *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, 2011, pp. 79–86.

[15] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996. [Online]. Available: http://dx.doi.org/10.1109/2.485845

[16] G.-J. Ahn and R. Sandhu, "Role-based authorization constraints specification," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 207–226, Nov. 2000. [Online]. Available: http://doi.acm.org/10.1145/382912.382913

[17] A. H. Karp, H. Haury, and M. H. Davis, "From abac to zbac: the evolution of access control models," *Hewlett-Packard Development Company, LP*, vol. 21, 2009.

[18] N. Dan, S. Hua-Ji, C. Yuan, and G. Jia-Hu, "Attribute based access control (abac)-based cross-domain access control in service-oriented architecture (soa)," in *Computer Science Service System (CSSS), 2012 International Conference on*, 2012, pp. 1405–1408.

[19] S. Hai-Bo, "A semantic- and attribute-based framework for web services access control," in *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*, 2010, pp. 1–4.

[20] X. Zhang, S. Oh, and R. Sandhu, "Pbdm: a flexible delegation model in rbac," in *Proceedings of the eighth ACM symposium on Access control models and technologies*, ser. SACMAT '03. New York, NY, USA: ACM, 2003, pp. 149–157. [Online]. Available: http://doi.acm.org/10.1145/775412.775431

[21] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "SAVI testbed: Control and management of converged virtual ICT resources," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, Ghent, Belgium, 2013, pp. 664–667.

[22] O. Foundation, "Openstack keystone architecture," http://docs.openstack.org/developer/keystone/.

[23] E. Halili, *Apache JMeter*. Packt Publishing, 2008.