2013 IEEE Sixth International Conference on Cloud Computing

Data On-boarding in Federated Storage Clouds

Gil Vernik*, Alexandra Shulman-Peleg*, Sebastian Dippl†, Ciro Formisano‡, Michael C. Jaeger†,
Elliot K. Kolodner*, Massimo Villari§
* IBM Haifa Research Lab
† Siemens Corporate Research and Technologies
‡ Engineering Ingegneria Informatica SPA
§ University of Messina

***Abstract*—One of the main obstacles hindering wider adoption of storage cloud services is vendor lock-in, a situation in which large amounts of data that are placed in one storage system can not be migrated to another vendor, e.g., due to time and cost considerations. To prevent this situation we present an advanced on-boarding federation mechanism, enabling a cloud to add a special federation layer to efficiently import data from other storage clouds. This is achieved without being dependent on any special function from the other clouds.**

We design a generic, modular on-boarding architecture and demonstrate its implementation as part of a VISION Cloud, which is a large scale storage cloud designed for content-centric data. Our system is capable of integrating storage data from various clouds, providing a common global view of storage data. The users can access the data through the new cloud provider immediately after the setup, maintaining the normal operation of applications, so that they do not need to wait for the completion of the data migration process. Finally, we analyze the payment models of existing storage clouds, showing that transferring the data via on-boarding federation with a direct link between clouds can lead to significant time and cost savings.

1 Introduction

Cloud platforms should fulfill the requirements for scalability and flexibility, allowing rapidly redeploying and moving resources. This is achievable for compute resources, but it is not common practice for storage. Existing storage clouds still do not allow true data mobility and cannot easily migrate their data across providers. The work "Above the Clouds" of Armbrust et al. [1], named the problem of "vendor lock-in" of the stored data to be the second among top ten obstacles for growth in Cloud Computing. The authors named the lack of standardized storage access APIs as one reason. Today, the Cloud Data Management Interface standard from the SNIA (CDMI [2]) exists. However, it lacks adoption by the larger cloud storage providers. In addition, applications generate so much data today that the resulting transfer time could require longer interruptions to services.

There are companies and products that have specialized in moving data (e.g., Nasuni, Racemi, Gladinet) or providing a common view of data (e.g., IBM CastIron). The main business case for such products is either the migration from a classic IT infrastructure to a cloud offering, or the prevention of vendor lock-in. However, third party tools cannot fully leverage the underlying storage cloud platforms for faster and more transparent migration.

Our goal is to prevent vendor lock-in by introducing a special federation layer as part of the storage cloud infrastructure. Our approach covers three areas: (1) standard API and interoperability; (2) efficient and transparent data migration; and (3) system security and access control. To cover the first issue, we adhere to the CDMI standard, allowing for interoperability between CDMI-compliant storage providers. Second, we introduce the concept of *on-boarding federation*, allowing an enterprise to move its data from one storage cloud provider to another (e.g., due to economical, legal or functional considerations) while providing continuous access and a unified view over the data in the old cloud and the new cloud, and over data in transit. Our approach forms a federation between the clouds. The data is migrated by a background process without interrupting services. The third area is the access control architecture, which targets the federation of two autonomous access control systems protecting the data in the two clouds. It is important to note that we ensure data consistency and completeness without introducing any centralized components or requiring any modifications to the old cloud. This preserves the benefits of distribution and scalability, and also makes our architecture suitable for future deployments with other storage cloud systems.

Here, we present an implementation over a VISION Cloud [3] system, which is an EU-funded project for a scalable and federated storage system providing content-centric access to its storage. In contrast to public cloud offerings such as Amazon S3 and Windows Azure Blob Storage, or specific hardware appliances, VISION Cloud stresses support for rich metadata as an integral part of the storage. As part of this project, partners also develop use case applications for telecommunications services, media production systems, healthcare services and enterprise applications.

The work is organized as follows. In Section 2, we describe our federation architecture and its implementation over the VISION Cloud. Section 3 discusses the security issues, comparing several access control models; Section 4 describes their implementation. Section 5 presents the overhead of fed-

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257019.

978-0-7695-5028-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CLOUD.2013.54

eration as well as the access control models. Section 6 covers economic issues, payment models and their benefits, while in Sections 7 and 8 we present related work and conclude.

2 Federation Architecture and Implementation

We start by presenting the general architecture of the VISION Cloud project [3] which builds a storage cloud comprised of tens of geographically dispersed data centers (DCs), where each DC may contain thousands of compute and storage nodes across multiple clusters. The cloud can serve millions of clients with billions of objects. These data objects contain data of arbitrary type and size, as well as rich metadata describing their content and handling. They are grouped in containers, which provide context and are used for the purposes of data management, isolation, and placement (i.e., a container is the minimal unit of placement and a container replica cannot be split across clusters). Containers may also have rich metadata associated with them, which effects how they and their objects are managed. The VISION Cloud account model consists of tenants and users, where a tenant may represent an enterprise or an organization that subscribes to cloud storage services. Tenant administrators are able to create unique user accounts, allowing their users to authenticate with the VISION Cloud system.

Here, we present the on-boarding federation system, which is implemented as part of VISION Cloud but can be easily added to other cloud systems wishing to make available to their customers a service for migration (on-boarding) from other providers. The new provider (New Cloud) is responsible for moving the customer data to itself from the old cloud provider (Old Cloud). Furthermore, following the federation set up, applications and users begin accessing their data through the New Cloud, which provides access to customer data remaining in the Old Cloud. Thus, the applications that access the migrated data should not be influenced by on-boarding and should work transparently. The Old Cloud is assumed to be unaware of the on-boarding and is not required to introduce any modifications.

Federation of storage clouds requires selecting the appropriate level of data granularity and abstraction. Our approach adopts the concept of a cloud storage container as the basic unit of federation. Establishing containers as the units of management for federation has many advantages. Containers are a well established concept in cloud storage environments, even across providers. In VISION Cloud each container belongs to a well defined tenant, hence the existing authentication and authorization infrastructure can be reused. In addition, using the container as a federation unit, allows applications to manipulate objects in federated storage containers as they would in containers in the normal unfederated case.

To activate an on-boarding federation, there is a need to "link" containers in the New Cloud to containers in the Old Cloud. To persist the linkage information, it is stored as part of the system metadata of the container in the New Cloud. This allows the New Cloud to always know which local containers are "linked" to other containers in the Old Cloud and which

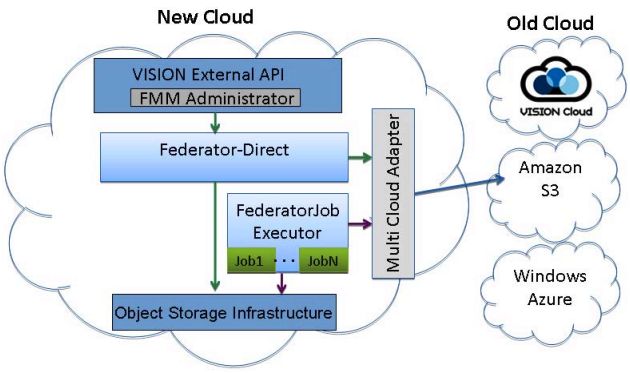

Fig. 1. High Level Federation Components

data objects should be on-boarded. We call such a pair of "linked" containers (one in the New Cloud to one in the Old Cloud) a *federated container*. Upon completion of the container "linkage" process, clients may start working with the container from the New Cloud immediately and there is no need to wait until all objects are on-boarded to the New Cloud. The New Cloud is responsible to provide users access to storage that has not yet been on-boarded from the Old Cloud, thereby providing the customer a unified view of the storage/data that resides in the New and in the Old Cloud. Our federation model also assumes that after the on-boarding setup the users work with the New Cloud only. All on-boarding software components run in the New Cloud; the Old Cloud does not need to provide any special functionality.

To foster scalability, all of the federation components are implemented in a stateless way and all the information is persistently stored as part of container metadata in the underlying distributed object store. Having no central instance controlling the federation allows preserving the cloud's robustness and durability. For example, the cloud can continue operating as usual, moving and relocating the containers for its internal purposes. Since all containers, including their metadata are replicated across the clouds's clusters, a federation will neither be corrupted by these management operations nor by disk or communication failures. Furthermore, an application using federation management is not dependent on a specific instance of the management service, and the configuration can be continued from a different instance of it.

2.1 On-boarding federation components

Below we describe the functions that are required for on-boarding and present the modular components implementing them (see Figure 1). They expose clear interfaces that allow their implementation and integration above additional, non VISION Cloud systems in the future.

Federation Administration: The *Federation Management and Monitoring Administrator (FMM)* module, provides federation services to other VISION Cloud components (e.g., for internal backup) as well as to external clients. FMM implements a RESTful API to manage federation with CRUD (create/read/update/delete) operations over standard HTTP

commands. As detailed in Table I, each federation is assigned a unique id that is used for its management.

Service	HTTP Method	URL
Create	PUT	/{tenant}/federations/federation
Read	GET	/{tenant}/federations/{id}/
Update	POST	/{tenant}/federations/{id}/federation
Delete	DELETE	/{tenant}/federations/{id}/federation
List	GET	/{tenant}/federations/

TABLE I
RESTFUL API FOR FEDERATION MANAGEMENT

In addition, FMM offers a monitoring and statistics service providing an overview of the on-boarding federation process. FMM calculates several performance and time indicators for an ongoing federation including: estimated on-boarding complete time, average transfer speeds, transfer progress and on-boarding history.

Federation triggering, direct execution and unified view: The *FederatorDirect* component is responsible for providing a unified view of a container distributed over the New and the Old clouds. It is interposed before the underlying object store, acting as an in/out filter for client requests, examining all external requests before passing them on to the object store for execution. When a request arrives, FederatorDirect inspects the container's system metadata to determine if the container participates in an active on-boarding federation; this is indicated by special reserved metadata keys. If so, FederatorDirect takes action based on the request type and the object's status. For example, if the request is to read an object that is not yet on-boarded (i.e., still located in the Old Cloud), FederatorDirect will immediately on-board it into the New Cloud and will return its data to the client. If the request is to list a container's objects, FederatorDirect will return the details of all the objects regardless of their physical location, i.e., by merging the lists from the Old and New Clouds.

Federation background execution: The *FederatorJobExecuter* generates *FederatorJobs*, which move data from the Old Cloud to the New Cloud in the background in a fast, efficient and decoupled way requiring no direct user or application interaction. The FederatorJobExecutor generates multiple jobs for each on-boarding relationship, assigning each job a list of objects that it needs to on-board from the Old Cloud. When a job becomes active, it processes this list and on-boards the objects. Each successfully on-boarded object is removed from the list. In case of a failure, a job resubmits itself back to the FederatorJobExecuter, which is responsible to generate, manage, execute, schedule and equally distribute FederatorJobs across the cluster. An important consideration is how to execute on-boarding without interfering with the normal operation of the clouds. For example, the FederatorJobExecuter can be configured to execute FederatorJobs only when the network load is low.

Interoperability with other Clouds: Various cloud providers expose differing APIs, so that the Old Cloud may have a different API than the New Cloud. To achieve interoperability with various Old Clouds, the *Multi-Cloud Adapter* contains implementations of multiple existing cloud APIs, e.g., S3, Swift, etc. In addition to the simple implementation of APIs this module also allows converting metadata (including system metadata) from the format supported by the Old Cloud to that of the New Cloud. Furthermore, we provide a framework allowing to easily implement and plug-in a new access driver for a new storage cloud API. The adapter is used by both the FederatorDirect and the FederatorJob components to access the New and the Old Clouds, thereby providing flexibility in the federation configuration.

2.2 On-boarding federation flow

On-boarding federation consists of two stages: the setup and the execution. To initiate a new federation, a container owner uses the API of the FMM to provide the details of the containers in the two clouds that should be federated. FMM uses the object storage infrastructure of the New Cloud to persist the federation configuration as part of container metadata, which is carefully protected with dedicated ACLs. Following the initiation of the federation by the FMM, execution passes to FederatorDirect, which inspects the container metadata to identify whether it contains special reserved keys that indicate an activation of on-boarding federation. When FederatorDirect identifies that a new federation setup request has arrived, it performs the following tasks: (1) verifies that it can access the relevant containers in both clouds; (2) sends a GET request to the container in the Old Cloud and obtains a list of objects to be on-boarded; and (3) requests the FederatorJobExecuter to generate FederatorJobs and distribute them across the cluster, assigning each job a subset of objects to transfer.

As soon as the federation is correctly set up and active, the FederatorJobs start copying the contents of the remote container into the local container and write status information to the metadata of the local container, so that the on-boarding process can be tracked by the Federation Management component. FederatorDirect "listens" to all requests coming to the New Cloud, trying to resolve them. If the data object is already on-boarded, it just forwards the request back to the normal get-object-flow and the object is returned to the requesting instance. If not, FederatorDirect initiates an immediate copy of the requested object from the remote container to the local container and then transfers the object to the client. In case of a list request for the objects of a container, FederatorDirect also provides a unified list of the local and remote containers.

To illustrate the flows, we detail examples of user operations on a federated container: *container_new* in the New Cloud and *conatiner_old* in the Old Cloud. Assume that *container_old* contains three objects: $\{obj1, obj2, obj3\}$, where only *obj1* was on-boarded to *container_new* so far.

Container list operation: A client sends a list operation for *container_new*. FederatorDirect intercepts this request, and sends it to the object storage layer on the New Cloud as well as the Old Cloud to obtain a list of objects in *container_old*. Having both results, FederatorDirect merges them into a single response comprised of the objects $\{obj1, obj2, obj3\}$, even though $obj2$ and $obj3$ are not yet on-boarded.

Object read operation: A client sends a GET request for *container_new/obj2*. Since *obj2* is not yet on-boarded, the object storage layer of the New Cloud responds with a object-not-found message. Based on this response, FederatorDirect reads the *container_new* metadata. Since the metadata shows an active on-boarding federation, FederatorDirect sends a GET *container_new/obj2* operation to the Old Cloud. Upon receiving *obj2*, FederatorDirect stores it in the New Cloud then returns it to the client.

Object metadata update: A client sends a request to update the metadata of *container_new/obj3*. Since $obj3$ is not yet on-boarded, FederatorDirect will try to on-board it, as in the object read example above. After the object is on-boarded, its metadata will be updated in the New Cloud, which will always contain the most updated object version.

3 Access control architectures

To enable on-boarding the federation components of the New Cloud need the permissions necessary to read the data from the Old Cloud and write the data to the New Cloud. Below we discuss the relevant access control architectures.

- *Shared IAMs:* In this set up the two clouds use the same internal Identity and Access Management systems (IAM), e.g., two object storage clouds deployed on the same public infrastructure, such as OpenStack.
- *External IAM:* In this case each cloud customer (tenant) has its own IAM, defining its identity and access domain, that establishes a trust relationship with both clouds. This allows a user to authenticate with a single set of credentials for applications running in various cloud and non-cloud systems. Commonly this is supported via Single Sign-On (SSO) mechanisms, such as SAML or OpenID, depicted in Figure 2.
- *Two IAMs:* Each cloud may have its own, internal or external IAM. Existing SSO mechanisms are insufficient to support this architecture and there is a need to introduce additional components as discussed below.

When implementing an on-boarding scheme there is a need to specify the identities used to retrieve the data. Here, we describe the identities that can be used to gain access to the Old Cloud, but the same approaches are applicable to authenticate with the New Cloud as well.

Setting up a special on-boarding user: While setting up the on-boarding process, a special, artificial on-boarding account (*user_onboard*) is created on the Old Cloud. The credentials of this account are passed to the New Cloud and are used in the authentication process. Furthermore, *user_onboard* is granted temporary read access to the container in the Old Cloud. This is achieved by updating the ACLs or the relevant access control policies of the Old Cloud, which may be time consuming and also requires revocation when on-boarding is complete. Since reusing the same *user_onboard* for multiple on-boarding requests may violate the principle of least privilege, it is recommended to generate a special on-boarding account for each federation setup, deleting it right after the corresponding on-boarding completes.

On-boarding on the behalf of the end user: This approach uses the identity of the end user (e.g., resource owner) and delegates his access rights to the New Cloud. If properly implemented, this may allow confining the access rights of the New Cloud, thereby preserving the security of the system. However, it burdens the end user by requiring his involvement in the delegation process.

Access Control Flow	**On-boarding identity**	**Advantages**	**Disadvantages**
Credential passing	On-boarding user	Protocol simplicity	Credential exposure
Bearer tokens	On-boarding or end user	Token simplicity	Tokens may be stolen; requires end user involvement
Full delegation	End user	Confinement of access rights	Protocol complexity; requires end user involvement

TABLE II
COMPARISON OF ACCESS CONTROL APPROACHES

Regardless of the identity scheme chosen, there is a need to specify the authentication protocol and its flow. For example, *user_onboard* may authenticate via protocols like HTTP Basic/Digest or a token based scheme. Below we present three relevant flows that are compared in Table II.

Credential passing: To allow authentication of on-boarding requests, the credentials of an account in the Old Cloud are passed to the New Cloud. Since passing the end user credentials is highly insecure, it is recommended to use the credentials of the temporary *user_onboard* created for this purpose. The main advantage of this scheme is its simplicity both from the user and cloud provider perspective. Its drawbacks are the security of the credentials which may be stolen and misused.

Bearer Tokens: In this setup the user approaches the IAM of the Old Cloud, authenticates, and asks for a long term token, which he passes to the New Cloud to perform the on-boarding. Since anyone possessing a bearer token can use it to gain access, it should be carefully protected (e.g., via encryption). It is recommended to scope the token reducing its access rights as much as possible. If the underlying protocol does not support this, it may be better to authenticate as *user_onboard* obtaining a token with limited access rights.

Full Delegation Technique: In this scenario, the *end user* is in charge of activating the on-boarding procedure. The user authorizes the system to act on his behalf, potentially with reduced access rights. Figure 3 illustrates that unlike regular SSO protocols, this solution does not require having a shared IAM and established trust relationships with both clouds. Each cloud may work with a different IAM, either external or internal. The end user delegates his access rights to the New Cloud, specifically to the processes doing the on-boarding. This can be achieved by enhancing standard WEB SSO solutions with delegation protocols like OAuth or SAML V2.0 Condition for Delegation (which we refer to as *SAML2Delegate*). Among these we adopted the latter, due

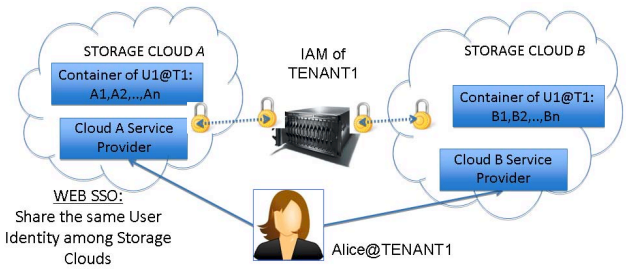

Fig. 2. Web Single Sign On (SSO) for accessing different Web Services with a *unified* User Identity.

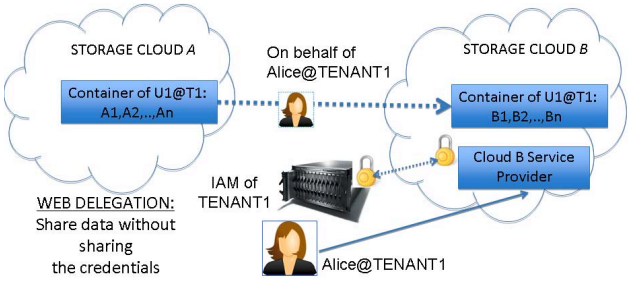

Fig. 3. Web User Delegation for accessing user's resources with a *Consumer (Cloud Storage B)* application.

the flexibly and security of the OASIS suit of Web Services Security (WSS) protocols to which SAML belongs.

Table II compares the approaches described above. As seen in the table, increasing the security by employing delegation techniques comes at the price of simplicity and usability.

4 Access Control Implementation in VISION

VISION Cloud implements a modular access control infrastructure supporting multiple authentication methods, including the three on-boarding access control flows described above. Below we describe the VISION Cloud access control implementation and its support for a SAML-based flow. Section 5 further compares this solution with additional authentication protocols used by VISION Cloud.

The access control architecture of the VISION Cloud is based on the cooperation of two models: Attribute Based Access Control (*ABAC*), used to check users privileges based on a set of attribute-based policies, and Access Control Lists (*ACLs*) that are attached to stored resources and protect them. VISION Identity and Policies Services (*IPS*) provide services for authentication, user management and ABAC-based policy management. VISION Cloud supports Identity Federation by integrating IPS with SAML SSO based on Shibboleth. This allows federating VISION identities with an IAM suppporting SAML, thereby allowing the federation of a VISION Cloud with another VISION Cloud or any external domain that supports SAML. For example, in the scenario described in this paper, the Old and the New Clouds have their own IAM servers, each of which defines an identity domain.

To support the architectures described above, termed *External IAM* and *Two IAMs*, we employ identity federation together with a *forwarding model*. In this model the New Cloud takes the identity of the user (through impersonation) and forwards the SAML assertion provided by the user. This assertion is used as a bearer token to acquire the privileges of the user in the Old Cloud. The flow of this model includes the following stages, depicted in Figure 4. (1) The user invokes the GetAssertion service to obtain a SAML authentication assertion. The Service Provider (SP) intercepts this call and discovers the External Domain of the request. (2) The user is redirected to the External Domain IAM to provide it valid credentials and then back to the GetAssertion service. (3) The user invokes the GetAssertion service to obtain an assertion identifier. (4) This identifier is passed to the VISION Federator (FederatorDirect or FederatorJobExecuter). (5) The VISION Federator inserts the assertion identifier in the HTTP headers of the requests to retrieve data. Based on the identifier, the VISION IPS of the Old Cloud gets the actual assertion stored by the SP and validates it by checking its signature, lifetime and attributes.

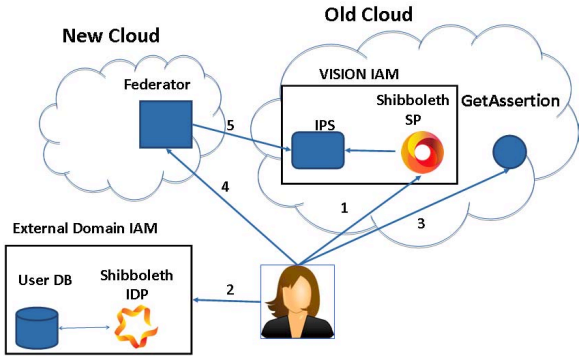

Fig. 4. Forwarding model applied in VISION

This model is based on bearer tokens, and a more secure solution is a full delegation in which the end user delegates only a subset of his rights. *SAML2Delegate* extends the SAML language to support this flow. To obtain a *delegation* from the user, the New Cloud, obtains a new assertion from the IAM, which contains an identity of the New Cloud listed as a *delegate* of the *end user*. As before, this assertion is sent to the Old Cloud with the actual request. This model ensures that there is no privilege elevation and the delegated access rights are less than or equal to the access rights of the end user.

5 Performance Analysis

In this section we present some performance results. First, we estimate the slowdown during normal operation for accesses to a federated container during on-boarding. The slowdown occurs for object misses, i.e., on an access to an object that was not yet on-boarded to the New Cloud. In this case, there is a delay until FederatorDirect brings this object from the Old Cloud. To measure this overhead we disabled the federation background processes in a New Cloud and created

two containers as follows: a container (with 100 objects) in the Old Cloud and a new empty container in the New Cloud. We then linked the container in the New Cloud to the container in the Old Cloud. To measure the costs of an object miss, we sent 100 consecutive requests to the New Cloud, such that each request to the New Cloud causes an object miss. Each such GET request triggered FederatorDirect to on-board the corresponding object from the Old Cloud. These 100 requests to the New Cloud led to the on-boarding of the entire container. After the on-boarding completed, we sent the same 100 GET requests to the New Cloud requesting the same objects, which were now present in the New Cloud. Notably, the average overhead for 100 GET operations with object misses was 24%, while the average overhead for listing the objects of the federated container (when all of the objects were still in the the remote container) was 22%. The overhead for GET was measured for objects of size 1K and 1M, for which the network transfer time is not the bottleneck. Obviously, when the object size increases, the network becomes the dominating factor, but the overhead is always lower than the 200% overhead of the naive currently available solution where a client manually downloads the objects from the Old Cloud and then uploads them to the New Cloud. One of the central advantages of the proposed on-boarding federation solution is that much of the overhead for normal operation (e.g., for object misses) can be made invisible to the user by on-boarding the objects in a background job, executed during hours of low load.

Next, we analyze the performance of access control methodologies and their influence on federation. While protocols like SAML enhance system security, they may add performance overhead. Here, we measured this overhead and compared it to less secure protocols, trying to find out whether the price paid for the added security is acceptable in the context of on-boarding federation. In addition to the SAML-based solution described in Section 4, we evaluated HTTP Basic and the UUID bearer token model of OpenStack Keystone. Table III summarizes the authentication overhead, which is the extra time consumed by 100 GET operations with authentication compared to those without authentication, as measured by the ApacheBench (ab) tool. Obviously, this overhead becomes less noticeable when the object size is increased from 1K to 100M byte. Furthermore, due to differences in the potential deployments it is difficult to make a fair comparison between the protocols and our goal is only to estimate their general applicability for on-boarding. As can be seen the overhead of all the schemes is reasonable, where in our setup HTTP basic was the fastest and the Keystone UUID was the slowest (due to the fact that when caching is disabled the UUID scheme contacts Keystone for the validation of each token). Notably, the SAML protocol gave a reasonable overhead, showing that it is applicable to the on-boarding federation scenario and can be safely used to increase its security.

Object size (B)	HTTP Basic (%)	SAML (%)	KeyStone UUID (%)
1K	8.65	14.84	17.96
1M	7.41	13.24	16.40
10M	4.55	7.39	12.55
100M	2.88	3.34	6.62

TABLE III
OVERHEAD OF AUTHENTICATION PROTOCOLS

6 PAYMENT MODELS

To estimate the benefits of our proposed solution for on-boarding, we evaluate the overall cost to transfer storage data out of an Old Cloud. Notice that storage costs can be an important factor, since a customer continues to pay the Old Cloud for the storage until on-boarding completes and the data is removed from the Old Cloud. We consider the two common approaches for exporting data from a cloud: via network transfer and via media shipment services offered by cloud providers.

On-boarding costs via network When on-boarding over the network, the total cost of on-boarding will be sum of the bandwidth cost, the request cost and the storage cost. Bandwidth cost is the cost to transfer a given volume of storage out of the cloud over the network. Request cost is the cost paid for the requests to GET (read) the storage. Storage cost is the cost for storing a given capacity of storage. Notice that all of these costs may be tiered, so that a higher cost is paid for the first bytes (or requests) and a lower cost for the last bytes (or requests).

On-boarding costs via export on media Storage cloud providers typically also provide the capability to receive data stored on the cloud on physical media mailed to the customer. Here the costs comprise the storage media cost, the export time cost, the media delivery costs, and as before the storage cost, since the customer will continue to pay for the data in the Old Cloud until it is on-boarded to the New Cloud and deleted from the Old Cloud.

On-boarding overhead We define the *on-boarding overhead* to be the total cost for on-boarding including the storage price for the on-boarding period, divided by the storage cost for one month for the amount of storage on-boarded. A higher on-boarding overhead points to a lower incentive for transferring the storage to another cloud.

To illustrate the costs, we calculate the costs for onboarding data out of Amazon S3. We base our calculations on S3's April 2013 pricing for the US Standard Region and where possible use the Amazon S3 Cost Calculator. We assume that the average object size is 1MB, and we calculate the number of operations for on-boarding accordingly. We consider storage costs of a single day as a minimal charge unit.

Figure 5 shows the on-boarding overhead for various data transfer options as a function of the amount of storage to be on-boarded. We show costs for transfer over an Internet connection at average rates of 512Mbps and 1Gbps, export on disks and export using Amazon Direct Connect, which is a direct high bandwidth connection. For Amazon Direct

This section relies on pricing information for Amazon S3 that is publicly available as of April 2013. Actual costs may be affected by several factors, including rounding errors and assumptions on the AWS cost model made in this paper.

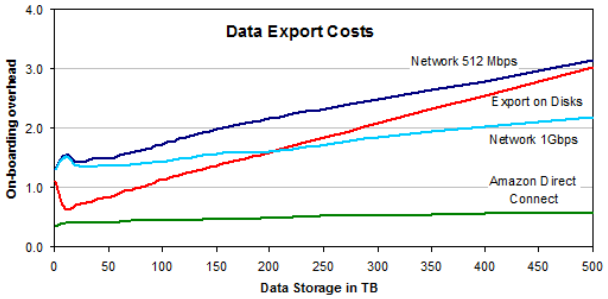

Fig. 5. Costs of data transfer out of S3

Connect, there is a charge for provisioning the port and a charge for the amount of data transferred. In our calculation we assume that the 10Gbps port can be provisioned just for the time of the transfer and that the port's transfer rate can be fully utilized, which may not be realistic. For export on disks we assume a disk size of 2TB, the eSATA interface type and expedited shipping. Using these parameters the Amazon Cost Calculator projects an average transfer speed to disk of 85MB/sec.

An obvious conclusion from calculations shown in Figure 5 is that the total time consumed by on-boarding is a major factor in the overall cost. In particular, a longer on-boarding time leads to higher storage costs that the customer must continue to pay to the Old Cloud. Thus, data should be on-boarded at the highest possible bandwidth. Furthermore, the on-boarding overhead for 100TB for all of the options is low enough such that if the New Cloud provides a clear advantage over the Old Cloud the cost to switch clouds is not too high (and the New Cloud could even provide an incentive). However, the on-boarding overhead for 500TB is significantly higher, so that only the high bandwidth network options are feasible.

The on-boarding approach presented in this paper, allows transferring the data directly between the clouds, thus significantly reducing the transfer time, and thus reducing the overall cost. If the New Cloud can maintain a direct high bandwidth connection to the Old Cloud, this will additionally reduce the cost.

7 Related Work

As Sheth and Larson explain in their work about federated databases [4] the term federation was introduced already in 1979 and in 1985 Heimbigner and McLeod summarized this concept as follows [5]: "A federated architecture represents a balance between two conflicting requirements: the autonomy of the entities versus a reasonable degree of coordination." There are four basic capabilities that characterize each entity of the federation, which "keeps authority about the information passed to the other entities of the federation" and "has authority to create a global view of the data that is available among all the entities of the federation". An entity of the federation "is not forced to perform tasks of another entity of the federation" and "can autonomously decide to enter or leave the federation". This basic characterization holds also for the approach presented in this paper.

Here, we distinguish between two types of federation: multi-layered and on-boarding. In contrast to on-boarding, which does not assume any special cloud topology, layered federation, is usually managed from an additional component that is not part of the federated clouds. Examples of such systems were presented by Tordsson et al. [6], Kurze et al. [7] as well as the recent Colony system, that federated several Openstack Swift deployments in a Colony by introducing a Swift Dispatcher component, prefixing the container names. Unlike this work, we do not assume any naming conventions or architectural components common to the two clouds.

Up to now, little work addresses on-boarding federation, which in addition to data management and migration, also require solving the problems of access control and common APIs, which we review below.

Multi-cloud APIs: Federation at the API level is a prerequisite for on-boarding, in particular, a common interface to transparently use services regardless of their underlying interface and implementation, decoupling implementations and services. The Cloud Data Management Interface (CDMI) lead by SNIA is a recent standardization effort for Web-based APIs for storage clouds [2]. The VISION Cloud project adopts and promotes this effort.

However, even in the absence of common APIs, it is still possible to federate between several different clouds, by implementing adapters supporting their corresponding APIs. Examples of such adapters are provided by companies like Deltacloud and RightScale, as well as by open source projects like *jclouds* and *libcloud*, among which we adopted the latter in our implementation.

Cloud access control: Cloud has not kept pace with the enormous volumes of user identities that network administrators must manage and secure. An identity fabric, linking multiple applications to a single identity was proposed to address this problem [8]. Recently, Chadwick and Casenove [9] described the security APIs allowing to grant federated access, fine grained access control and delegation, integrating their approach with the Eucalyptus S3 Service.

Interoperability in federated heterogeneous cloud environments is addressed in [10], proposing a model to delegate trust between trustworthy parties satisfying certain constrains. Pearson at al. [11] also introduce a privacy manager in order to care for data compliance according to the laws of different jurisdictions. Huang et al. [12] introduced a SAML-based Identity Federation Broker for service clouds. A recent work, FACIUS [13], describes the use of SAML for non Web-based services, presenting an implementation of SAML with SSH. They report important evaluations with respect to requirements, performance, security, and legal aspects. Interestingly, their measurements show an almost 300% overhead of adding SAML (SAML-enabled SSH login takes 1.01 seconds, while regular SSH only 0.24).

A recent trend for managing credentials and delegations is

based on leveraging the PKI technology with X.509 certificates. For example, combining X.509-based $libabac$ package with attribute-based access control and an authorization policy language called "role-based trust" Schwab and Faber [14] provided a trust framework suitable for federated community clouds. Another example, is the recent adoption of PKI by the OpenStack Keystone component [15]. These projects use technologies that are comparable with those used in our paper, however, they do not address the specific identity management problems that occur in on-boarding federation.

Addressing the issue of delegation, an AAuth system [16] enhanced the OAuth capabilities using the encryption in attribute-based access control system exploiting metadata. Alternatively, we address the issue of delegation by leveraging the well established SAML protocol and its *SAML2Delegate* extension, which increases system security with a reasonable performance overhead (see Section 5).

Payment models: According to the view of the VISION Cloud project, Cloud federation may represent a compelling business model for SMEs, where many stockholders (i.e., Cloud providers, tenants and customers) interact with each other for creating new opportunities and satisfying even more needs. Relevant work is presented in [17], where the authors are able to estimate what the revenues are when services are outsourced in federated clouds. The authors introduced a formula able to capture *Costs* and *Profits*. The federation described in [17] is aimed at computation management (consolidation of VMs among Clouds). It shows similarities to the work done by Celesti et al. [18] and Rochwerger et al. [19], where the federation problems for data management between several IaaSs are addressed. In the latter case the authors described how to elastically enlarge the physical resource of an IaaS in a transparent way. Unlike this work, we address a different pricing model, which takes into account the amount of data and the speed of its transfer.

8 Conclusions and future work

In this paper we presented the concept of on-boarding federation for storage clouds. We described and implemented an on-boarding federation architecture over the VISION Cloud infrastructure, showing that small objects can be on-boarded with only a 24% overhead. Addressing security issues, we discussed several access control architectures, evaluating their suitability for on-boarding. We implemented and compared several access control protocols, analyzing their influence on performance. We observed that even well established protocols like SAML can be safely used to increase system security, while adding a reasonable overhead. When analyzing the existing solutions to transfer data between clouds, we came to a conclusion that the cost for the client to do on-boarding on its own is high, and that there is a need for direct cloud to cloud on-boarding. Since the proposed architecture, does not require any changes in the infrastructure of the Old Cloud, it can be used to on-board data from existing clouds like Amazon S3. We introduced the notion of on-boarding overhead and used it to show that large amounts of data with low network bandwidth leads to stickiness of the data at the storage provider.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[2] "Cloud Data Management Interface Version 1.0.2," SNIA Storage Networking Industry Association, Jun 2012. [Online]. Available: http://snia.org/sites/default/files/CDMIv1.0.2.pdf

[3] E. K. Kolodner, *et al.*, "A cloud environment for data-intensive storage services," in *CloudCom*, 2011, pp. 357–366.

[4] A. Sheth and J. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Computing Surveys*, vol. 22, no. 3, pp. 183–236, 1990.

[5] D. Heimbigner and D. McLeod, "A federated architecture for information management." *ACM Trans. Inf. Syst.*, vol. 3, no. 3, pp. 253–278, 1985.

[6] J. Tordssona, R. S. Monterob, R. Moreno-Vozmedianob, and I. M. Llorente, "Optimized placement of a computational cluster across multiple clouds."

[7] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud federation," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*. IARIA, Sep 2011.

[8] E. Olden, "Architecting a cloud-scale identity fabric," *Computer*, vol. 44, no. 3, pp. 52–59, Mar 2011.

[9] D. W. Chadwick and M. Casenove, "Security apis for my private cloud - granting access to anyone, from anywhere at any time," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 792–798.

[10] W. Li and L. Ping, "Trust model to enhance security and interoperability of cloud environment," in *Cloud Computing*, Nov 2009, pp. 69–79.

[11] S. Pearson, Y. Shen, and M. Mowbray, "A privacy manager for cloud computing," in *Cloud Computing*, Nov 2009, pp. 90–106.

[12] 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*, May 2010, pp. 115–120.

[13] J. Kohler, S. Labitzke, M. Simon, M. Nussbaumer, and H. Hartenstein, "Facius: An easy-to-deploy saml-based approach to federate non web-based services," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, Jun 2012, pp. 557–564.

[14] J. Chase and P. Jaipuria, "Managing identity and authorization for community clouds," Department of Computer Science, Duke University, Tech. Rep., 2012, technical Report CS-2012-08.

[15] "Welcome to keystone, the openstack identity service!" September, 2012. [Online]. Available: http://docs.openstack.org/developer/keystone/

[16] A. Tassanaviboon and G. Gong, "Oauth and abe based authorization in semi-trusted cloud computing: aauth," in *Proceedings of the second international workshop on Data intensive computing in the clouds*, ser. DataCloud-SC '11. New York, NY, USA: ACM, 2011, pp. 41–50.

[17] I. Goiri, J. Guitart, and J. Torres, "Characterizing cloud federation for enhancing providers' profit," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, Jul 2010, pp. 123–130.

[18] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Integration of clever clouds with third party software systems through a rest web service interface," in *Proceedings - IEEE Symposium on Computers and Communications*, 2012, pp. 827–832.

[19] B. Rochwerger, S. Naqvi, C. Ponsard, J. Latanicki, P. Massonet, and M. Villari, "A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures," in *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 1510–1517.