An Approach for Hybrid Clouds Using VISION Cloud Federation

Uwe Hohenstein, Michael C. Jaeger, and Sebastian Dippl
Corporate Technology, Siemens AG
Munich, Germany
Email: {Uwe.Hohenstein, Michael.C.Jaeger, Sebastian.Dippl}@siemens.com

Enver Bahar
Corporate Technology, Siemens AG
Munich, Germany
Email: bahar.enver@gmail.com

Gil Vernik and Elliot K. Kolodner
IBM Research - Haifa
Haifa, Israel
Email: {gilv, kolodner}@il.ibm.com

Abstract—**Hybrid clouds combine the benefits of a public cloud and on-premise deployments of cloud solutions. One scenario for the use of hybrid clouds is privacy: since public clouds are considered unsafe, sensitive data is often kept on premise. However, other non-sensitive resources can be deployed in a public cloud in order to benefit from elasticity, fast provisioning, or lower cost. In this work, we present an approach for hybrid cloud object storage based on the federation of storage resources. It uses the metadata of the objects and containers as a fundamental concept to set up and manage a hybrid cloud. Our approach extends an existing scheme for implementing federation for object storage developed by the VISION Cloud project.**

***Keywords*-Cloud storage;hybrid;federation.**

I. Introduction

Cloud storage refers to the broader term of cloud computing that represents a novel provisioning paradigm for resources. The National Institute of Standards and Technology (NIST) defines "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1].

Cloud storage as one specific cloud service applies the major characteristics of cloud computing to storage, which are a) virtually unlimited storage space, b) no upfront commitment for investments into hardware and software licenses, and c) pay per use [2]. Depending on the viewpoint, a couple of other characteristics may also become relevant, e.g., support for multi-tenancy in order to serve multiple customers at the same time.

Moreover, the term cloud storage covers a wider area than provisioning approaches and models; cloud storage refers to software implementing storage services. Most notably, *Not only SQL* (NoSQL) database servers are a specific cloud storage solution that gained popularity recently in this context [3]. The understanding about NoSQL databases is that corresponding database servers or services follow a different approach than the traditional table-model provided by relational database servers, implied by an adaptation to distributed systems and cloud computing environments, e.g., by relaxing the Atomicity, Consistency, Isolation, Durability (ACID) characteristics of the traditional (relational) database servers by means of *Basic Availability, Soft state, Eventual consistency* [4] (BASE).

A lot of cloud storage solutions such as NoSQL databases can be used even in a non-cloud computing manner. In such a setup, a storage server can be deployed similar to a traditional database server managed on premise. Of course, the advantages of cloud storage are then partially lost: virtually unlimited storage space is limited by the storage hardware provided by the own platform; investments for such hardware must be taken. The software must be set up and the setup must be planned in advance in contrast to flexible provisioning and pay-as-you-go models. Therefore, such a deployment makes sense, if a software system has to be installed locally while taking advantage of specific characteristics, e.g., of NoSQL technology. In fact, there are a number of motivations for keeping critical data on premise, on private servers rather than utilizing public cloud storage offerings:

- Data storage cannot be delegated because of regulatory certifications. For example, data store that contains legally relevant material could be subject to possible confiscation and thus provisioning of such data cannot be delegated.
- There are often privacy constraints. For example, a data store that holds employees' invention disclosures before being submitted to patent offices might not be suitable to be placed at a public cloud provider.
- A cloud provider offers a certain Service Level Agreements (SLA) or reliability, which is insufficient. In this case, a private on-premise proprietary storage solution may be the choice for keeping critical data.

As pointed out above, the disadvantage of using private on-premise storage solutions opposed to public cloud storage is obvious. In particular, it is likely less flexible and potentially more expensive in terms of cost, since an on-premise solution typically cannot always benefit from the same economy of scale that can be achieved by a public provider.

In this work, we show how to benefit from both worlds by integrating on premise storage services for critical data with public cloud storage service for non-critical data. Through this approach, flexibility and potential cost advantages as well as high SLA requirements can be achieved as required for the individual data entities.

We present our metadata-based approach for a hybrid cloud based on the cloud storage federation scheme developed by the European funded VISION Cloud project [5][6]. VISION Cloud aims at developing next generation technologies for cloud object stores including content centric access to storage. It offers first-class support for metadata for the storage entities, i.e., objects and containers, and enables management functionality based on such metadata, for example, describing and managing federation through container and object metadata.

In order to achieve a hybrid scenario, combining public and private storage, we also use container and object metadata to describe the federation setup. Such an approach to describing federation provides a unified and location-independent access interface, i.e., transparency for data sources, while leaving the federation participants autonomous.

The remainder of this work is structured as follows: Section II explains the VISION Cloud software that is relevant and is used for this work: the concept, particularly of using metadata, the storage interfaces, and the storage architecture. The hybrid cloud approach of VISION Cloud is then presented in Section III. We explain our approach, particularly the architectural setup in Section IV, and continue in Section V with further useful federation scenarios. Section VI is concerned with related work. A brief evaluation of this approach is explained in Section VII. This work ends with Section VIII providing conclusions and future work.

II. The VISION Cloud Project

The EU project VISION Cloud [5] is developing a cloud storage system that allows for the efficient storage of different types of content. The approach supports storage for objects (videos, etc.) together with metadata describing their content. An increasing number and a variety of applications exist that envisage a growing need for such an object store. New media authoring applications are currently arising with video files such as Ultra High-Definition and 4K resolutions. Archives for virtual file systems of virtual machines occur in the area of virtualization and cloud computing. And finally, mobile users use their smart phones for producing and capturing multimedia content in an exponentially growing manner [7].

These all are examples where we expect an increasing number of large and unstructured storage objects. The VISION Cloud project intends to provide an object storage system that is capable of handling large objects and files. Another goal of VISION is the ability to easily ingest content of different types, to analyze the content and enrich its metadata, and to smoothly access the content through a variety of end user devices. This functionality extends the plain data storage features offered by today's cloud storage providers.

A. The VISION Concept of Metadata

A common way to handle content is to put it into files and to organize it in a hierarchical structure. This enables navigating the hierarchy in order to finally find a particular item. However, it becomes more and more difficult to set up an appropriate hierarchy that provides flexible search options with acceptable access performance and intuitive categories for ever increasing amounts of data.

Thus, the target of the VISION Cloud project is to provide an appropriate basis for organizing objects including content-centric access facilities. In contrast to public cloud offerings such as Amazon S3, Microsoft Blob Service, or specific hardware appliances, VISION Cloud puts emphasis on supporting metadata flexibly and making metadata an integral part of the storage system [5][6]. Moreover, VISION Cloud supports private cloud installations and does not require any specialized hardware.

The approach of content-centric storage (CCS) does not restrict the user to organizing his content in hierarchies. Rather, the user describes the content through metadata allowing him to access the content based on its associated metadata. Moreover, an additional layer of the storage system derives metadata from usage statistics or access mechanisms. The basis for this approach to content-centric storage are efficient mechanisms to automatically create or ingest and retrieve metadata about the content. Then, this is the entry point to objects.

To illustrate the intention and scope, consider YouTube as an example: a video (i.e., an object) on YouTube can possess a set of metadata associated with it, metadata of different characteristics. Some metadata remains static (e.g., the uploader of a video), other is dynamic (for instance, the number of views). Some metadata is related to content (e.g., categories applied to the video) and other is of technical nature (e.g., the resolution of the video). YouTube enables one to access the video by searching metadata. Hence, a user can ask for a video that is 'most viewed' or that belongs to the category 'drama' [8].

VISION Cloud offers similar functionality in a more general form. In particular, it extends the scope to any type of data, no matter whether videos, text, pictures, or documents in any form and from any source. While handling and using metadata in a conventional data object storage system is typically restricted to storing and retrieving metadata together with the object, VISION Cloud provides the ability to update and append new metadata to objects as well as the ability to find objects based on their metadata values and their relationships to other objects.

B. Interface

The basic data model of VISION Cloud is similar to the Cloud Data Management Interface (CDMI) [9] standard interface for access and distinguishes between containers and objects. CDMI defines a standardized way to store objects in a cloud. The standard covers create, retrieve, update, and delete (CRUD) operations by defining simple requests with the HyperText Transfer Protocol (HTTP) according to the REpresentational State Transfer (REST) principle. In this model, the underlying storage space is abstracted and primarily exposed using the notion of a container.

Containers are similar to buckets in other storage solutions. A container is not only a useful abstraction for storage space, but also serves as a grouping of the data stored in it. The storage creates, retrieves, updates, and deletes each object as a separate resource. Containers might be organized in a hierarchical manner. The following REST examples give an impression about the CDMI-based interface of VISION: `PUT /CCS/MyTenant/MyContainer` creates a new container for a specific tenant $MyTenant$. Then, `PUT /CCS/MyTenant/MyContainer/MyObject` can be used store an object into this container.

The metadata is passed as content or payload of a HTTP PUT request. The first question arises how to distinguish the container from an object. By the CDMI standard, this is solved by using HTTP header fields indicating a data type for this request. A full request for creating a new container thus looks as in Figure 1:

```
Example: PUT /CCS/MyTenant/MyContainer
X-CDMI-Specification-Version: 1.0
Content-Type: application/cdmi-container
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Accept: application/cdmi-container
{   metadata : { key1 : value1, key2 : value2 }   }
```

Figure 1: HTTP PUT Request

III. Hybrid Cloud Approach

The goal of cloud federation is to provide a unified and location-independent access interface, i.e., a transparency of data sources in various clouds of different providers, while leaving the federation participants autonomous. Thus, creating a federation of existing clouds supports the client with a unified and combined view of storage and data services across several providers and systems. There are several approaches to provide federation:

1) The first one is to put a new federation layer on top of the clouds to be federated. This means that every access to the federation must be passed to this federation layer. The federation layer might offer additional services such as distributed queries. In any case, each of the federation members remains accessible by its own interface.
2) Instead of introducing a new layer, each federated cloud can be an access point to the federation, i.e., can accept requests. If the cloud itself is unable to answer the request, it delegates the request or parts of it to respective clouds.
3) Controlling clouds and storage location can be part of a language in the sense of a multi-database approach [10]. That is, operations can be performed on storage locations that are explicitly specified, e.g., by means of wildcards expressions.

In the VISION Cloud project, approach (2) is pursued [11]. A federation is accessible from any cloud in the federation. The development of the VISION Cloud project mainly provides two mechanisms that can be used as a base for federations:

1) *Object storage with use of metadata:* In VISION Cloud, all objects are allowed to contain user-definable metadata items that can be used in several ways to query for objects inside the cloud storage system. It is possible to employ a schema for these metadata items to enforce the existence of certain metadata fields and hence enforcing a certain structure.
2) *Adapters for storage clouds:* VISION Cloud includes an additional layer for integrating cloud storage systems, e.g a blob storage service of some larger public cloud offering.
3) *A fine granular Access Control List (ACL) infrastructure:* VISION Cloud was designed with security in mind and provides fine granular access control that are attachable to tenants, containers and objects. If containers are taking part in a federation, the administrator of a federation can provide new ACLs for objects that are being moved in a federation.

Based on the existing work carried out by the VISION Cloud project, this work uses the concept of federation to define a hybrid cloud setup for leaving critical data on premise, while using the metadata processing facilities of the CCS functions of VISION Cloud. The approach consists of two main parts: the administration and setup of a hybrid cloud and its operation.

A. A Hybrid Cloud Setup Based on VISION Cloud Federations

First, there is an administrator for the setup. The administrator is responsible for creating and maintaining a federation of two cloud systems. We propose a way to introduce the clouds to each other by a data structure that contains *metadata* (rather than the data itself) about remote data. In VISION Cloud, a federation is defined between two data containers. In order to create a federation, the administrator has to send a data structure describing the federation to one of the containers taking part in the federation. A typical federation administration data structure in VISION Cloud, for example for an Amazon S3 member, uses the payload [11] in Figure 2:

```
"federationinfo": {
 // information about target cloud
 "eu.visioncloud.federation.status": "0",
 "eu.visioncloud.federation.job_start_time": "1381393258.3",
 "eu.visioncloud.federation.target_cloud_type": "S3",
 "eu.visioncloud.federation.target_container_name":
    "example_S3_bucket",
 "eu.visioncloud.federation.target_region": "EU_WEST",
 "eu.visioncloud.federation.type": "sharding",
 "eu.visioncloud.federation.is_active": "true",
 "eu.visioncloud.federation.local_cloud_port": "80",
 // credentials to access target cloud
 "eu.visioncloud.federation.target_s3_access_key":
   "AKIAIHSZSHAHVWZEAZTGWJOBQ",
 "eu.visioncloud.federation.target_s3_secret_key":
   "2glBUIdO3qQUTLoCeBxTrYoxYzqgV5A2us/Hcd+p",
 "eu.visioncloud.federation.status_time": "1381393337.72"   }
```

Figure 2: Sample Payload

This specification mainly describes the data required for accessing a member's cloud storage. This structure allows one to store some specific information about the clouds such as public/private cloud URIs, container names in both private and public containers, etc. Such a specification creates a link between the clouds and enables certain tasks as data sharding and querying among them. For hybrid scenarios, the federation type can now be chosen as `sharding`. Upon completion of the container linkage process, both private and public containers become aware of each other.

To manage federation, a REST Service is available in VISION Cloud providing the basic CRUD operations (create / read / update / delete) to administer federation instances over standard HTTP commands and to handle these structures. PUT creates a new federation instance by passing an id (in the Uniform Resource Identifier (URI) and the federation info in the body. GET gives access to a specific federation instance and returns the federation progress or statistical data. A federation specification can be deleted by DELETE. Finally, all federation instances can be listed with `GET /{tenant}/federations/`. The result will be an array of federation URIs. For details please refer to the project deliverable [12].

This concept of federation is also applied to the creation of the hybrid setup placing a similar request, although the implementation is different as explained later.

B. Operation of the Hybrid Cloud Setup

The second role involved in an hybrid cloud scenario is the client who wants to access the data in the new cloud system. The client now is provided with a unified view of the data that resides in both the private and public cloud and becomes ready to shard data among the clouds. Any CRUD operation will work on both of the clouds, and all sharding operations are completely abstracted from the client. The operations of the hybrid cloud setup is completely transparent for the client of a container, and the client might even not be aware where the data he accesses resides.

IV. Architecture

Our main focus is put on hybrid clouds, i.e., using public and private clouds at the same time for an application. The goal is to provide a uniform access to several autonomous clouds each hosting a cloud store. Moreover, we want to benefit from the recent VISION Cloud architecture as much as possible.

In general, a hybrid cloud has to tackle heterogeneity of the units to be combined. In the context of storage federation, there are several types of heterogeneity. At first and most obvious, each cloud provider such as Amazon, HP, IBM, or Microsoft has management concepts and Application Programming Interfaces (APIs) of its own, which are currently proprietary, despite some emerging standards, e.g., CDMI [9]. And then at the next lower level, the federation has to take into account the heterogeneity of data models of the cloud providers.

The hybrid cloud should provide an abstraction over the individual storage clouds. This means on the one hand that a unified interface is offered for all the clouds and thus supporting query features without knowing what data is available in what cloud. On the other hand, the hybrid cloud should provide means to control the placement of storage items. Since we here focus on distributing data over private and public clouds according to their confidentiality, each item (object) must have a confidentiality level associated with it in order to allow for storing it in a corresponding cloud.

In fact, the implementation of the content-centric storage service (CCS) of VISION Cloud helps to handle heterogeneity by allowing us to wrap homogeneous units, each with a CCS interface. An instance of a VISION Cloud CCS sits on top of a single storage system. However, multiple underlying storage system types are supported by CCS due to an adapter architecture that accommodates multiple storage interfaces. Currently CCS adapters are available for the proprietary VISION Cloud storage service, Amazon S3, CouchDB, MongoDB, and the Windows Azure Blob storage. That is, the CCS architecture supports multiple cloud providers, as long as a storage adapter is provided.

The CCS directly connects to a storage server's IP and port number, either referring to a single storage server or to a load balancer within a cluster implementation. If we put CCS on top of each storage server, the CCS would have to manage all the distribution, scalability, load balancing, and elasticity. This would tremendously increase the complexity of CCS. Moreover, CCS would re-implement features that are already available in numerous cloud storage implementations. All of the currently supported storage system types have a built-in cluster implementation. Among the CCS candidates, CouchDB has an elastic cluster implementation named BigCouch. MongoDB has various strategies for deploying clusters of MongoDB instances. And Windows Azure Blob storage, as well, is a distributed environment. To our knowledge and published material by the vendors, we can assume that these cloud systems are able to deal with millions of customers and tens of thousands of servers located in many data centers around the world.

Hence, our decision was to put CCS on top of these cluster solutions due to several benefits. CCS is just a bridge between the load balancer and the client. All scalability, elasticity, and partitioning is done by the storage system itself. Therefore, there is no need for CCS to deal with scalability, elasticity, replication, or duplication within the cloud. Since there is only a single CCS instance for each type of cloud storage, consistency issues are also handled internally.

A. Technical Implementation

The implementation of the federation service of VISION Cloud is not used for the hybrid cloud setup. Instead, the service has been technically implemented in the layer that provides the content-centric storage functions (the "CCS"). In fact, in order to enable the hybrid cloud in the CCS, several extensions have been made to the CCS: A new *ShardService* has been added to CCS the task of which is to intercept requests to the CCS and decide where to forward the request. The ShardService implements a reduced CDMI interface and plays the key role to shard in hybrid environments. As already mentioned, we give the right to the client to determine data confidentiality. This selection is done through the data creation process and a metadata item should indicate data confidentiality.

Having researched several sharding mechanisms, we chose to implement *Key Based Partitioning*. Key based partitioning provides a perfect match to our needs by using a metadata key to define the sharding strategy for separating public and private clouds. We enable clients to create some keywords to define the confidentiality such as *confidential*, for example. If there is such a keyword among the metadata, then this data will be directed to the private cloud. Otherwise, it will be directed to the public cloud. The impact of keywords on sharding the data can be specified in the federation.

B. Scenarios

The following examples should illustrate the approach. A simple PUT request mechanism works as follows in the ShardService:

1) The metadata of the object is checked for an item indicating confidentiality such as *confidential* : *true*.
2) Then the metadata of the container is fetched (not the object) to obtain the connection information for both clouds.
3) If the data is confidential, the private cloud's connection information is used. The ShardService connects to this cloud and forwards the request to it.
4) If data is not confidential, then the public cloud's connection information is used to send the request to this cloud.

A GET request through the ShardService behaves differently, since it does not necessarily contain a specification regarding which cloud contains the object. Therefore, every GET request is sent to all clouds participating in the federation:

1) As soon as the GET request arrives in the cloud, the container metadata is requested to gather the

connection information about the private and public clouds.

2) The ShardService connects to both clouds and run the same GET request for each of them in parallel.
3) The results of requests are combined and the result is sent back to the client.

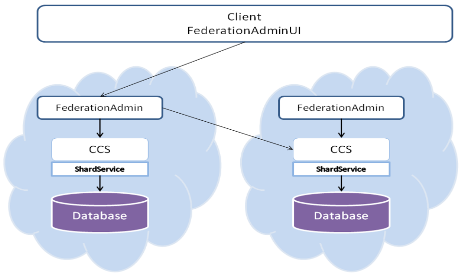

Figure 3: Federation creation

The entire key-based sharding principle can be explained with an example. Beforehand, we assume we have two different containers in two different clouds. Let us name these two containers as *vision1* and *vision2*. Additionally, our two clouds are addressed by these URLs respectively: *vision-tb-1.myserver.net* and *vision-tb-2.myserver.net*.

First, we need to make the two clouds aware of each other, to be more precise, we need to federate them. A sketch of the process can be seen in Figure 3. By using the newly introduced ShardService and its HTTP API, we do a PUT request with the payload of Figure 4.

```
http://cloud_url:cloudport/MyTenant/sharded/vision1
{  "target_cloud_url" : "vision-tb-2.myserver.net",
   "target_cloud_port" : "8080",
   "target_container_name" : "vision2",
   "local_container_name" : "vision1",
   "local_cloud_url" : "vision-tb-1.myserver.net",
   "local_cloud_port" : "8080",
   "type" : "sharding",
   "private_cloud" : "vision1",
   "public_cloud" : "vision2"   }
```

Figure 4: Federation payload

The above JSON string is a sample of our data component. We need to mention that a full dataset contains information regarding the private and public cloud types, urls, users, authorization information, etc. Upon completion of the request, the two clouds vision-tb-1 and vision-tb-2 enable sharding at the container level. From now on, *vision-tb-1* will be our private cloud and *vision-tb-2* will be our public cloud. Such a specification is needed for any container to act as a shard (cf., the local target_container_name).

The PUT request must also be sent to the second cloud, however, with an ”inverted” payload. This is done implicitly.

Since the two clouds are federated, we can now perform data CRUD operations in a sharded way. In order to store confidential data, we need to perform the request in Figure 5.

```
PUT vision-tb-1.cloudapp.net:8080/CCS/siemens
  /vision1/newObject
{  "confidential" : "true"  }
```

Figure 5: PUT request for storing confidential data

Figure 6 shows that the ShardService decides to store *newObject* in the *private* cloud which is *vision-tb-1*. Note that there is no need to indicate the access to the federation as part of the PUT request.

If the data is not confidential we can replace `"confidential" : "true"` with `"confidential" : "false"` in Figure 5 (see also Figure 7):

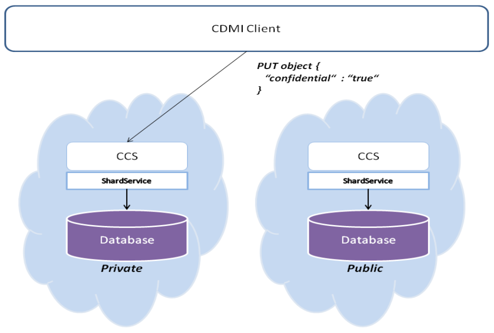

Figure 6: Storing confidential data.

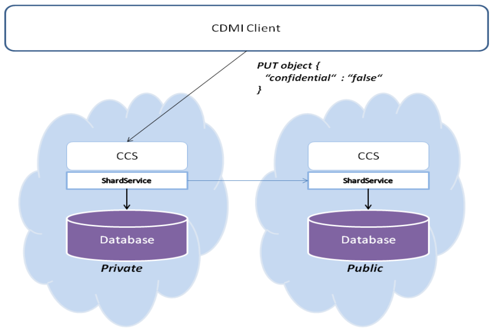

Figure 7: Storing non-confidential data.

Because of the metadata value, the ShardService will connect to the public cloud and the data will be stored in the public cloud, i.e., *vision-tb-2*. It is also possible to submit a such PUT requests to *vision-tb-2*.

There is no additional interface to which object creation operations and queries need to be submitted. Figure 6 and 7 show that a request can be sent to any shard in any cloud, and that cloud passes the request to the proper cloud to handle the request. This principle can be extended to handle several public or private clouds as well.

V. Further Hybrid Cloud Scenarios

So far, we have considered a hybrid cloud scenario where the location of objects is determined according to metadata. Being able to work with several cloud storage systems in a sharded manner offers several advantages. The first one is data confidentiality. Clients can store critical data on a secure cloud object store, and other data can be stored on general public cloud object stores, which might be cheaper and offer better extensibility.

However, the approach is more flexible and can be used in other scenarios as well. One scenario that VISION Cloud has implemented is a so-called on-boarding federation [11]. The purpose of this scenario is to migrate data from one cloud storage system to another. One important feature of the implemented on-boarding federation is to allow accessing *all* the data in the target cloud while the migration is in progress, i.e., while data is being transferred in the background. After the administrator has configured the federation, the objects will be transferred in the background to the client' container in the new cloud. If the client lists the contents of a federated container, all objects from all containers in the federation will be shown. If the client accesses objects which have not been on-boarded yet, the objects will be fetched on demand. The on-boarding handler intercepts GET-requests from the client and redirects them to the remote system on demand and schedules the background jobs for copying the clients' data from the remote cloud.

This helps to let a client become independent of a single cloud storage provider, i.e., a vendor lock-in is avoided. Having only a single vendor as a cloud storage service might limit the availability and scalability to that provider. In fact, the vendor lock-in of the stored data is the second among top ten obstacles for growth in cloud computing [2].

In the VISION Cloud project several further types of federation (cf. `federation.type` in Figure 4) have been discussed, but are not yet implemented: policy-based, synchronization, backup, multi-site-access, and company-merges. Based on the sharding mechanism presented in this paper, we can offer general strategies that distribute data according to a partitioning scheme. The idea behind sharding is splitting the data between multiple storage systems and ensuring that access to the data always occurs at the right place. In our case, we can partition and query data among the cloud storage systems.

The main advantage of database sharding is scalability, the ability to grow in a linear fashion as more servers are included to the system. Additionally, several smaller data stores are easier to handle and manage than huge ones. Furthermore, each shard contains less data and thus can be faster to query. Another positive effect is that each shard has a server of its own, resulting in less competition on resources such as CPU, memory, and disk I/O. And finally, there will be an increase of availability: If one shard fails, other shards are still available and accessible.

On the other side of the coin, several factors need to be considered to ensure an effective sharding. Although most of the applications are fault-tolerant, the storage tier is always the most critical part for reliability. Due to the distributed approach of multiple shards, the importance is even greater. To ensure a fault tolerant sharding precautions such as automated backups and several live copies of the shards need to be made. Next, a good partitioning scheme is required to match the needs of the system [13][14]; the following general approaches exist and can be controlled in our approach in principle:

1) *Vertical Partitioning*: All data related to a specific feature will be stored in the same place, i.e., images are in an image storage and videos in a video storage. On the one hand, this approach is easy to implement and causes only little overhead for the application. But on the other hand, the partitions might become uneven. Some shards might require more than one server.
2) *Range Based Partitioning*: In situations where data of a single feature needs to be distributed, it is important to find a meaningful way to spread the data. One approach is to define ranges for data values and to distribute data accordingly. Although this is also easy to implement, it will cause some unbalanced load distribution between the shards.
3) *Key Based Partitioning*: This requires a special entity with a unique key; this entity clusters data and can be used to identify the shard.
4) *Hash Based Partitioning*: Hash sharding involves processing values through a hash function to choose which server to store. Thus, each server will have a hash value, and each computed hash value will end up on one of those servers.
5) *Directory Based Partitioning*: This scheme keeps a lookup table somewhere in the cluster which keeps track of which data are stored in which storage. This approach means that the user can add servers to the system without the need of changing the application.

Further scenarios can support fault tolerance and high availability features in way that metadata control the number of replications. Those use cases will be part of future work.

VI. RELATED WORK

Though federation in cloud environments is still a research topic, some of the basic concepts and architectures of federation have already been researched intensively within the area of federated database management systems [10]. Sheth and Larson define a federated database system as a "collection of cooperating but autonomous component database systems" including a "software that provides controlled and coordinated manipulation of the component database system". This definition places the federated database layer outside and on top of the component database systems that make up the federation. [10] also introduces a possible characterization of systems along the dimensions of distribution, heterogeneity and autonomy, and differentiates between tightly coupled systems (where administrators create and maintain a federation) and loosely coupled systems (where users create and maintain a federation). Moreover, the authors describe a five-layer reference architecture for federated database systems.

One project that offers a unified API between several data stores is presented by Bunch et al. [15]. In this work, the authors present a single API from Google App Engine to access different open source distributed database technologies. Such a unified API represents a fundamental building block for working with cloud storage as well as local NoSQL database servers. In contrast to our solution based on CCS, the implementation described by the authors provides access only to a single storage system at a time.

For the concurrent use of different storage providers or systems, Abu Libdeh et al. [16] propose a cloud storage system which is named as Redundant Array of Cloud Storage (RACS). It is placed as a proxy tier on top of several cloud storage providers. The authors describe adapters for three different storage interfaces, and point out that it can easily be expanded to additional storage interfaces. The approach uses erasure coding and distributes the contents of a single PUT request across the participating storage providers. Therefore, such a (write) operation must wait till the slowest of the providers completes the request. This is in contrast to the sharding of our work, where a PUT request is routed to a single storage system.

Another work which seems to be close to ours is presented by Brantner et al. in [17]. They build a database system on top of Amazon's S3 cloud storage. We also support Amazon S3 as one of our storage layer options. In their future work, they intend to include support for multiple cloud storage providers.

Additionally, there are a lot of multi-cloud APIs or libraries enabling unified access to multiple different cloud storage systems; these include Apache Libcloud [18], Smestorage [19] and Deltacloud [20]. They provide unified access to different storage systems, and protect the user from API changes. Although they enable administration features like stopping and running the storage instances, their storage driver functionalities are restricted to basic CRUD methods, most of them omitting a query interface.

A notable concept is found in the area of Content Delivery Networks (CDN). A content delivery network forms a network of servers around the world which maintain copies of data. When a user accesses the storage, the CDN infrastructure delivers the website from the closest servers. According to Broberg et al. [21], current storage providers have emerged as a genuine alternative to CDNs. In their work, they describe a cheaper solution by using cloud storage with their Meta Content Delivery Network (Meta CDN). Although Meta CDN makes use of several cloud storage providers, it is not a storage system by definition, and mostly focuses on read performance and lacks write performance.

In addition to the work mentioned above, there are also hybrid cloud solutions. Most of the current hybrid cloud offerings provide data transfer from private to public – instead of providing a unified view. The first example is Nasuni [22], which is a primary storage option. It is a form of network attached storage, which moves the user's on-premise data to a cloud storage provider following encryption. Nasuni's hybrid cloud approach combines on-premise storage nodes that gather the data and encrypt the data. Then, they send the encrypted data to a public cloud, which can be hosted at Amazon Web Services or at Microsoft Azure. The user can either store all the data in a single public cloud store, or can distribute them over multiple stores. Nasuni implements a migration approach, rather than a sharding approach such as ours, since data is eventually moved to the public cloud.

Nimbula [23] is another company that provides a service allowing the migration of existing private cloud applications to the public cloud using an API that permits the management of all resources. CloudSwitch [24] has also developed a hybrid cloud that allows an application to migrate its data to a public cloud.

Nirvanix [25] is one of the companies offering a hybrid cloud option. They offer a private cloud on premises to their customers, and enable data transfer to the public Nirvanix Cloud Storage Network. Although it is a hybrid cloud, it forces one to use only Nirvanix products. This represents a vendor lock in when it comes to the selection of the public cloud. In contrast, our adapter approach is not limited to a specific public cloud service.

Hybrid cloud storage solutions in the marketplace today provide a range of offerings to meet different demands of customers. Although there are many such offerings, they pose the risk of a vendor lock-in, because most of the companies use their own infrastructure. The most suitable work that matches the approach of our work is MetaStorage [26]. It represents a federated cloud storage system that is able to integrate different cloud storage providers. MetaStorage implements a distributed hash table service that replicates data on top of diverse storage services. It provides a unified view between the participating storage services or nodes while implementing sharding between them.

VII. SIMULATION RESULTS

In this section, we provide some basic performance tests for our architecture. We have examined different cases in order to evaluate the query performance using the CCS on top of two clouds, private and public. Our aim is first to check performance on a hybrid cloud setup with an increasing number of requests, and second, to see how the performance changes with a multi-threaded implementation.

For the tests we used the same data sets and similar setup as in our former work [8][27]. The test data is acquired from Deutsche Welle, which is one of the partners of the VISION Cloud project. Deutsche Welle has an analysis application which crawls data from YouTube across a number of news channels. 90 channels a day were tracked for a given time-frame, in total 490.000 videos have been collected together with their metadata. For the evaluation of our work, we used a subset of this data, which has in total 46.413 videos. Each video has the same amount of metadata and exactly the same fields.

We used two machines with the configuration as specified in Table I. We located these two machines in the same geographical area, having the same network. This is sufficient to analyze the performance overhead of the architecture, however, does not give results about the overall performance, which depends on the latency anyway . Each machine had a CouchDB database, Tomcat application server, and our Java Web Archive components installed. The main rationale was to test how well the implementation scales with larger volumes. The tests show three different cases: one querying for videos published in one day, one for videos published over two weeks, and one for videos published over four weeks. By using each of the resulting video ids, we sent 91, 1990 and 2908 consecutive requests to the underlying storage. Apart from that we re-ran the tests with a Java Thread Pool implementation to see the effects of parallelism in our system.

At first we uploaded our sample dataset by using the sharding implementation. This resulted in storage of 23206 video metadata on one machine, and 23207 video metadata on another machine. Afterwards we queried both of the storage systems. The total stack ran 20 times, and the average values are taken. To increase the precision, the longest and shortest run times are excluded from the overall measurement before taking the average. The results can be seen in Table II. The setup and the test runs were the same as used in our previous publications.

TABLE I: MACHINES USED IN A HYBRID SETUP

Designation, Processor	**Cores**	**Clock Speed**	**L2 Cache**	**RAM**	**OS**	**Storage**
2-core (4 threads)	1/1	1.90Ghz	4MB	4GB	64bit Win7	128GB SSD
2-core	1/1	2.20Ghz	2MB	3GB	32bit Win7	250GB HD

The first column represents the test configuration, single threaded or thread pooled. The next column gives the resulting number of requests to the underlying storage system. The average of the measured times are given in milliseconds, and

TABLE II: RESULTS OF THE HYBRID SETUP

	Request Count	Average Total (msec)	Standard Deviation	95% Conf.	Single request
Single Thread	91	2852,7	216,9	134,4	31,69
	1990	62404,9	4086,0	2532,5	31,38
	2908	96826,8	3670,6	2275,0	33,31
Thread Pool	91	1014,6	67,1	41,6	11,27
	1990	25764,2	2617,3	1622,2	12,95
	2908	39252,6	2839,6	1759,9	13,50

single request times are calculated as an average time divided by the request counts. As it can be seen, the single request times do not change much as the number of requests increases and are on average 30 milliseconds, which is acceptable. Also of notice is the multi-threaded implementation. In all of the cases, it boosted performance significantly.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented a new approach for hybrid cloud storage that is based upon the idea of federation as carried out by the VISION Cloud project. Our approach provides a uniform interface for handling confidential and non-confidential data, the first kept in an on-premise data store, the later stored in a public cloud. The key idea is to use metadata for controlling where data is stored. As a technical basis, we use the VISION Cloud software stack [7] where such a metadata concept is an integral part. We show in detail how well-suited VISION Cloud and its storage system is to support hybrid scenarios and how to extend it in order to support hybrid scenarios.

In principle, it is possible to offer additional sharding scenarios in the VISION Cloud project, beyond the privacy scenario we have presented for this work. The overall approach also allows for adding various further sharding strategies, such as region-based, load balancing, or storage space balancing, redundancy level control, etc. In fact, our future work will be dedicated to extending the hybrid approach and to elaborating more on query load balancing (based on metadata). Another aspect that requires attention is migration if security settings are changing.

ACKNOWLEDGMENT

The research leading to the results presented in this paper has received funding from the European Union's Seventh Framework Programme (FP7 2007-2013) Project VISION-Cloud under grant agreement number 217019.

REFERENCES

[1] P. Mell and T. Grance, "The nist definition of cloud computing (draft)," NIST special publication, vol. 800, no. 145, 2011, p. 7.

[2] A. Fox et al., "Above the clouds: A berkeley view of cloud computing," Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, vol. 28, 2009.

[3] "Nosql databases," at: http://nosql-database.org, [retrieved: April, 2014].

[4] B. Tudorica and C. Bucur, "A comparison between several nosql databases with comments and notes," in Roedunet International Conference (RoEduNet), 2011 10th, 2011, pp. 1–5.

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

[6] E. Kolodner (2) et al., "Data intensive storage services on clouds: Limitations, challenges and enablers," in European Research Activities in Cloud Computing, D. Petcu and J. L. Vazquez-Poletti, Eds. Cambridge Scholars Publishing, 2012, pp. 68–96.

[7] "Vision cloud project consortium, high level architectural specification release 1.0, vision cloud project deliverable d10.2, june 2011." at: http://www.visioncloud.com, [retrieved: April, 2014].

[8] M. C. Jaeger et al., "Cloud-based content centric storage for large systems," in Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on, 2012, pp. 987–994.

[9] "Cloud data management interface version 1.0," at:http://snia.cloudfour.com/sites/default/files/CDMI_SNIA_Architecture_v1.0.pdf, year = 2010, month = April, note = [retrieved: April, 2014], SNIA Storage Networking Industry Association.

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

[11] G. Vernik et al., "Data on-boarding in federated storage clouds," in Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, ser. CLOUD '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 244–251. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2013.54

[12] "Vision cloud project consortium: Data access layer: Design and open specification release 2.0, deliverable d30.3b, sept 2012." at: http://www.visioncloud.com/, [retrieved: April, 2014].

[13] D. Obasanjo, "Building scalable databases: Pros and cons of various database sharding schemes," URL: http://www.25hoursaday.com/weblog /2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSchemes.aspx, vol. 10, 2009, p. 2012, [retrieved: April, 2014].

[14] R. Cattell, "Scalable sql and nosql data stores," ACM SIGMOD Record, vol. 39, no. 4, 2011, pp. 12–27.

[15] C. Bunch et al., "An evaluation of distributed datastores using the appscale cloud platform," in Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 305–312, [retrieved: April, 2014].

[16] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: a case for cloud storage diversity," in Proceedings of the 1st ACM symposium on Cloud computing, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 229–240, [retrieved: April, 2014].

[17] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska, "Building a database on s3," in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, p. 251.

[18] "Apache libcloud a unified interface to the cloud," at: http://libcloud.apache.org/, [retrieved: April, 2014].

[19] "smestorage," at: https://code.google.com/p/smestorage/, [retrieved: April, 2014].

[20] "deltacloud," at: http://deltacloud.apache.org/, [retrieved: April, 2014].

[21] J. Broberg, R. Buyya, and Z. Tari, "Service-oriented computing — icsoc 2008 workshops," G. Feuerlicht and W. Lamersdorf, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Creating a 'Cloud Storage' Mashup for High Performance, Low Cost Content Delivery, pp. 178–183.

[22] "Nasuni," at: http://www.nasuni.com/, [retrieved: April, 2014].

[23] "Nimbula," at: http://en.wikipedia.org/wiki/Nimbula, [retrieved: April, 2014].

[24] "Cloudswitch," at: http://www.cloudswitch.com/, [retrieved: April, 2014].

[25] "Nirvanix," at: http://www.nirvanix.com/products-services /cloudcomplete-hybrid-cloud-storage/index.aspx, [retrieved: April, 2014].

[26] D. Bermbach, M. Klems, S. Tai, and M. Menzel, "Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs," in Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, ser. CLOUD '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 452–459, [retrieved: April, 2014].

[27] M. C. Jaeger et al., "Extending cloud-based object storage with content centric services," in CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 8-10 May, 2013, 2013, pp. 279–289.