A Multi-criteria Approach for Large-object Cloud Storage

Uwe Hohenstein[1], Michael C. Jaeger[1] and Spyridon V. Gogouvitis[2]
[1]*Corporate Technology, Siemens AG Munich, Germany*
[2]*Mobility Management, Siemens AG Munich, Germany*

Keywords: Cloud Storage, Federation, Multi-criteria.

Abstract: In the area of storage, various services and products are available from several providers. Each product possesses particular advantages of its own. For example, some systems are offered as cloud services, while others can be installed on premises, some store redundantly to achieve high reliability while others are less reliable but cheaper. In order to benefit from the offerings at a broader scale, e.g., to use specific features in some cases while trying to reduce costs in others, a federation is beneficial to use several storage tools with their individual virtues in parallel in applications. The major task of a federation in this context is to handle the heterogeneity of involved systems. This work focuses on storing large objects, i.e., storage systems for videos, database archives, virtual machine images etc. A metadata-based approach is proposed that uses the metadata associated with objects and containers as a fundamental concept to set up and manage a federation and to control storage locations. The overall goal is to relieve applications from the burden to find appropriate storage systems. Here a multi-criteria approach comes into play. We show how to extend the object storage developed by the VISION Cloud project to support federation of various storage systems in the discussed sense.

1 INTRODUCTION

The National Institute of Standards and Technology (NIST) states that "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" (Mell and Grance, 2011). Hence, cloud computing represents a provisioning paradigm for resources in first place.

Cloud storage is certainly one important cloud resource that benefits from the major characteristics of cloud computing (Fox et al., 2009) such as

- virtually unlimited storage space,
- no upfront commitment for investments into hardware and software licenses, and
- pay per use for the occupied storage.

The term cloud storage is mostly associated with the recent technology of *Not only SQL* databases (NoSQL, 2017), which attained a lot of popularity. Implied by an adaptation to distributed systems and cloud computing environments, NoSQL databases follow a different approach than the traditional fix-schema based model provided by relational database servers. They heavily rely on distributing data across several computers and prefer a schema-less storage of data with a relaxed consistency concept. Certainly, the settled technology of relational databases - if deployed in a cloud - is also a kind of cloud storage. There are corresponding offerings from major Cloud providers. And finally, Blob stores represent a further type of storage that should be mentioned.

Hence, we notice an increasing heterogeneity of storage technologies even within the same category, with further differences from vendor to vendor, whether deployed on-premises or in the cloud, whether used as Platform-as-a-Service (PaaS) or as a special Virtual Machine on the IaaS level. Each individual cloud storage has virtues of its own.

To benefit at a broader scale, a combination of storage solutions seems to be useful. Some important aspects in this context are in the sense of polyglot persistence (Sandalage and Fowler, 2012):

- to use the most appropriate storage technology for each specific problem;
- to reduce costs by using a public cloud, by choosing an appropriate cloud provider, particularly under consideration of the various and complex price schemes and underlying factors such as price/GB, data transfer or number of transactions;

Hohenstein, U., Jaeger, M. and Gogouvitis, S.
A Multi-criteria Approach for Large-object Cloud Storage.
DOI: 10.5220/0006432100750086
In *Proceedings of the 6th International Conference on Data Science, Technology and Applications (DATA 2017)*, pages 75-86
ISBN: 978-989-758-255-4

- to take into account access time and latency, e.g., to use fast but expensive storage only when really needed but slow and cheap storage in other cases;
- to consider the differences between on-premises and cloud solutions with certain limitations, e.g., the maximal database size, a reduced features set;
- to use a hybrid cloud for security and confidentiality issues, i.e., keeping confidential data in a private cloud while taking benefit from the public cloud for non-confidential data;
- to shard data in general, e.g., according to geographic locations, costs etc.

These points are fully interleaved and demand for a multi-criteria approach. Therefore a model is needed that captures the necessary information and creates associations between all involved entities. This information can be used to find an optimal data placement solution for an overall benefit.

In this paper, we base our work upon a metadata-based approach for a cloud storage scheme developed by the European funded VISION Cloud project (Kolodner et al., 2011)(Kolodner (2) et al., 2012). VISION Cloud aimed at developing next generation technologies for storing large objects like videos and virtual machine images, accompanied by a content-centric access to storage. Following the CDMI proposal (CDMI, 2010), the approach relies upon objects and containers and offers first-class support for metadata for these storage entities.

VISION Cloud has implemented a simple federation approach that provides some basic access to several storage solutions. We extend the original federation approach to tackle the needs of a multi-criteria storage solution that attempts to combine different storage technologies. Indeed, having a federation, it is possible to benefit from the advantages of various storage solutions, private, on-premise and public clouds, access speed, best price offerings etc. while the same way avoiding the disadvantages of a single storage. A federation approach can provide 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 2 explains the VISION Cloud software that is relevant and extended in this work: the concept, especially of using metadata, the storage interfaces, and the storage architecture. The original cloud federation approach of VISION Cloud is also presented. We explain an extended federation approach, particularly the architectural setup, in Section 3, and continue with technical details in Section 4. Section 5 is concerned with related work. This work ends with Section 6 providing conclusions and future work.

2 THE VISION CLOUD PROJECT

VISION Cloud (Kolodner et al., 2011) was an EU co-funded project for the development of metadata-centric cloud storage solutions. The project developed a storage system and several domain applications where the handling of rich metadata provides new innovations. Domains targeted by VISION Cloud were telecommunication, broadcasting and media, health care, and IT application management.

These domains share the need for an appropriate object storage system. The telecommunication, the broadcasting, and media domains envisage the storage of videos, the health care domain application stores high resolutions diagnostics images, and the IT application management stores virtual machine images. They all share the need for growing storage capacity and large storage consumption per object. The images of virtual machines in a data center grow bigger. The media domain is moving to Ultra High-Definition and 4K resolution content. And in the telecommunication domain, sharing of video messages turns into a trend as the market share of high-resolution camera equipped handsets grows (VISION-Cloud, 2011). All these domains in VISION Cloud benefit from an object storage developed in the project and serve as a proof-of-concept: They require an increasing need of large capacity for the expectation to store a vast number of large objects and the ability to maintain rich metadata sets in order to navigate and retrieve stored content.

Pursuing a metadata-centric approach, VISION Cloud stresses the ability to represent the type and format of the stored objects. With such an awareness of the storage, functionality can be triggered depending on the execution context and the currently processed storage object. The automation based on the awareness also contributes to the ability to deal with a high number of objects stored in an autonomous manner.

2.1 The VISION Concept of Metadata

Classic approaches that handle large objects basically organize files in a hierarchical structure in order to allow navigating through the hierarchy and finally finding a particular item. However, it can be quite difficult to set up a hierarchy in an appropriate manner that provides flexible search options with acceptable access performance and intuitive categories for ever increasing amounts of data. Moreover, such a hierarchy has usually to be organized manually and is thus prone to outdated or wrongly applied placements. In addition, the problem arises how to maintain a hierarchy changes in a distributed environment.

One of the goals of VISION Cloud was to enhance the object storage with rich metadata handling capabilities. Looking at the cloud storage offerings that existed from the popular vendors at project start, VISION Cloud decided to focus on the storage and retrieval of objects *based on metadata* and the ability to perform autonomous actions on the storage node based on metadata (Kolodner et al., 2011).

The content can be accessed based upon using metadata. A lot of useful metadata is technical, such as the file format or the image compression algorithm. Looking at some video sharing plaforms, some obvious metadata is also already available such as the title, the author of the video, a description, the date of upload, or a rating provided by users who have watched the content. Of course, such (existing) metadata could be easily added during import, as one of the VISION Cloud demonstrators has shown (Jaeger et al., 2012). Besides objects, metadata can be also attached to containers, which hold several different objects. Using container metadata also enables storage handling information for the objects inside.

In addition to such types of metadata that requires just the import of objects, the VISION Cloud object storage has the ability to derive metadata from processing storage objects. VISION Cloud uses so called storlets (Kolodner (2) et al., 2012): Storlets are software modules, similar to stored procedures or triggers in traditional databases, which contain executable code to process uploaded storage objects. They can analyze storage object, e.g., deriving metadata to be attached to objects. Hence, one application of storlets is to run speech-to-text analyzers on video content in order to store the text resulting from the audio track as metadata. Then indexing can provide search terms as additional metadata attached to video content. Such an approach improves the ability to navigate across a large number of video objects drastically.

From a technical perspective, VISION Cloud uses a key-value objects tree in a dedicated storage for keeping the metadata only, along with a further basic object storage for the large objects themselves. It is the special ability of VISION Cloud to efficiently keep both the storage objects and their metadata in synchronization when considering the node-based architecture and envisaging horizontal scaling.

2.2 The VISION Content Centric Interface

Applications deal with metadata in VISION Cloud by using the so-called *Content Centric Interface* (CCI), an interface that maintains and allows for querying metadata. The content-centric approach relieves a user from establishing hierarchies in order to organize a high number of storage objects. VISION Cloud follows the Cloud Data Management Interface (CDMI, 2010) specification. CDMI standardizes the interface to object storage systems in general. However, the concept of the previously mentioned storlet, for example, is a CDMI extension not covered by the standard at the time of developing the VISION Cloud.

CDMI defines a standard for accessing and storing objects in a cloud specifying the typical CRUD (Create, Retrieve, Update, Delete) operations in a REST style (Fielding and Taylor, 2002) using HTTP(S). The user can organize storage objects using containers. Containers can be compared to the concept of buckets in other storage solutions.

In VISION Cloud, a container enables not only the organization of storage objects, it allows also to efficiently design queries and handle objects in general. 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*.
- `PUT /CCS/MyTenant/MyContainer/MyObject` then stores an object *MyObject* in this container.

The payload of a HTTP PUT request contains the metadata describing an object. To distinguish a container from an object, the type of data for this request is indicated in the HTTP Content-Type header field (`cdmi-container`). For example, a full request for creating a new container 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 : { content : video, format : mpeg3 }   }
```

Figure 1: HTTP PUT request.

2.3 VISION Cloud Architecture

The general VISION Cloud architecture was presented in previous work (Kolodner et al., 2011). Thus, we here focus on the storage and the metadata handling. Figure 2 outlines the structure of layering: The foundation is the object storage, which includes the ability to store and manage the metadata. For example, replication of storage objects with their metadata across nodes is handled by this layer. On top of this layer, every node has the Content Centric Service (CCS) deployed, which offers higher services such as a relationship concept. The Content Centric Storage implements the CCI. It extends CDMI by adding

some additional operations in order to provide rich metadata handling for applications. The application, depicted on the top of the two other layers in Figure 2, accesses the storage via the CCS.

Being a cloud service implementation, the VISION cloud storage software was designed to run on common appliance dimensions allowing one to deploy a massive number of parallel nodes in a data center in order to enable horizontal scaling. An application is principally able to access several nodes with CCS and underlying object storage stacks deployed. In fact, the distribution over nodes is transparent to the user and facilitated by load balancing in a VISION Cloud storage system.

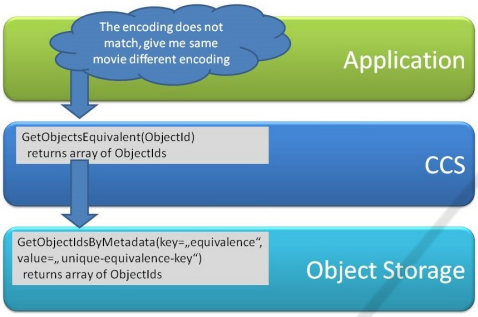

Figure 2: VISION architecture.

As an example, assume the application has a reference of an object and would like to find similar objects in the storage. This is a popular use case when it comes to different media encoding types for different end devices. The provider might store media content optimized for hand held devices or smart phones along with media content optimized for high-definition displays. The application uses the CCS to query for objects similar to the object already known. The similarity is a metadata feature that was provided especially as part of the ability to maintain relations between objects as metadata. The CCS defines how to encode relations by using key-value metadata on the object storage and decodes the relation-based query from the application into an internal metadata format which is not aware of relations. A corresponding query is sent to the underlying object storage.

The underlying object storage was developed as part of the VISION Cloud project. It provides a number of innovations in addition to the handling of metadata, e.g., execution of storlets or resiliency of storage items. If an application uses only the metadata handling features and not the storlets, i.e, only the basic features for storing and managing objects, the CCS can work also with other storage systems that are capable of storing metadata attached to objects and containers. The CCS uses an adapter concept to separate the integration of different storage servers. Therefore by implementing an adapter, other storage implementations can be integrated with CCS as well, given that the metadata handling requirements are provided in addition to a plain object storage. As part of the VISION Cloud project, the open source CouchDB document database was also integrated with the CCS.

Using a specific storage adapter, the CCS 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. Technically, the object storage and the CCS could reside on different machines or nodes. Also, a client could access the object storage directly, not using the CCS metadata handling capabilities. In the basic setup of VISION Cloud nodes, the CCS is deployed on every storage node, which is a (potentially virtual) server running a VISION Cloud node software stack. This avoids increased request response times resulting from the connections between different network nodes. Moreover, the CCS is capable of avoiding node management functionality and keeping track of the current status of object nodes. This enables a horizontal scaling of VISION Cloud storage nodes in general.

The request handling of CCS does not (need to) support sessions. As such, it can be easily deployed on the storage node as a module. The CCS can also work with storage system on different machines.

2.4 Basic VISION Cloud Support for Federations

Cloud federation aims at providing an access interface so that a transparency of data sources in different storage clouds of different provisioning modes is provided. At the same time it leaves the federation participants autonomous. Clients are able to leverage a federation with a unified view of storage and data services across several systems and providers.

In general, such a federation 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 has management concepts and Application Programming Interfaces (APIs) of its own, which may be proprietary or may implement industry specifications, e.g., (CDMI, 2010). And then at the next lower level, the federation has to take into account the heterogeneity of data models of the cloud providers. In fact, the implementation of the content-centric storage service of VISION Cloud helps to handle heterogeneity by means of adapters, thus allowing one to wrap heterogeneous units, each with a CCS interface. An instance of a VISION Cloud CCS sits on top of a single storage system. Moreover, the CCS architecture

supports multiple cloud providers and underlying storage system types, as long as a storage adapter is provided. Currently, CCS adapters are available for the proprietary VISION Cloud storage service or CouchDB.

The CCS communicates with the object storage using an IP connection. This means that the CCS can have several storage servers beneath with a load balancer in front. Hence, the VISION Cloud decision was to put CCS on top of these (homogeneous) cluster solutions due to several benefits. CCS is just a bridge between the load balancer and the client. All scalability, elasticity, replication, duplication, and partitioning is done by the storage system itself. Therefore, there is no need for CCS to re-implement features that are already available in numerous cloud storage implementations. In fact, current storage system types usually have a built-in cluster implementation already. For instance, the open source project CouchDB has an elastic cluster implementation named BigCouch. MongoDB as another open source example, has various strategies for deploying clusters of MongoDB instances. 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, the CCS does not have to manage all the distribution, scalability, load balancing, and elasticity. This would have tremendously increased the complexity of CCS.

The basic federation functionality of VISION Cloud allows the CCS to actually distribute requests between multiple storage nodes. Depending on metadata, the CCS can route storage requests to different storage services. This appears similar to the features of a load balancer. Although the CSS does not represent a load balancer, the role of the CCS in the software stack could be useful to service similar purpose: In fact, the CCS can distribute requests over different storage nodes not based on the classic load balancing algorithms for distributing load, but based on quality characteristics and provided capabilities that are matched with the available storage clouds.

Moreover, VISION Cloud was designed with security in mind and provides fine granular access control lists (ACLs). ACLs are attachable to tenants, containers, and objects.

2.5 Use of Federations

The basic VISION Cloud federations features have been used and slightly extended in two federation scenarios. The approaches are briefly presented now.

2.5.1 On-boarding Federation

The first scenario is a so-called on-boarding federation (Vernik et al., 2013). The purpose of this scenario is to migrate data from one storage system to another, the *target*. One important feature of the on-boarding federation is to allow accessing *all* the data via the target cloud while the migration is in progress, i.e., while data is being transferred in the background. The on-boarding scenario helps to avoid a vendor lock-in, which is the second among top ten obstacles for growth in cloud computing according to (Fox et al., 2009). With on-boarding being enabled to move data without operational downtime, a client becomes independent of a single cloud storage provider.

To set up on-boarding, an administrator has to create and maintain a federation of the two involved storage systems. A federation is always defined between two containers: The administrator has to send a ”federation configuration”, which describes the federation to the target container, because the target container will initiate a pull mechanism.

```
"federationinfo": {
 // information about remote cloud
 "eu.visioncloud.federation.status": "0",
 "eu.visioncloud.federation.job_start_time": "1381393258.3",
 "eu.visioncloud.federation.remote_cloud_type": "S3",
 "eu.visioncloud.federation.remote_container_name":
    "example_S3_bucket",
 "eu.visioncloud.federation.remote_region": "EU_WEST",
 "eu.visioncloud.federation.type": "sharding",
 "eu.visioncloud.federation.is_active": "true",
 "eu.visioncloud.federation.local_cloud_port": "80",
 // credentials to access remote cloud
 "eu.visioncloud.federation.remote_s3_access_key":
   "AFAIK3344key",
 "eu.visioncloud.federation.remote_s3_secret_key":
   "TGIF5566key",
 "eu.visioncloud.federation.status_time": "1381393337.72" }
```

Figure 3: Sample payload.

The payload in Figure 3 describes a typical federation configuration in VISION Cloud. The data is required for accessing a member’s cloud storage, i.e., the remote storage to be moved to the target cloud, here for an Amazon S3 member. With this configuration request, a link between the clouds is created.

A new REST service, the *federation service*, provides the basic CRUD operations to configure and handle federations. This federation service is deployed along with the CCS. PUT creates a new federation instance by passing an id in the Uniform Resource Identifier (URI) and the federation configuration in the body. GET gives access to a specific federation instance and returns the federation progress or statistical data. A federation configuration can be

deleted by DELETE. For details please refer to the project deliverable (VISION-Cloud, 2012).

After the administrator has configured the federation, the objects of federated containers will be transferred in the background to the container in the target cloud. If a client asks for the contents of the container, all objects from all containers in the federation will be returned. Thus, objects that have not been on-boarded yet will be fetched from the remote source, too.

A special on-boarding handler of the federation service intercepts GET-requests from the client and redirects them to the remote system for non-transferred containers and schedules the background jobs for copying data from the remote cloud.

2.5.2 Hybrid Cloud

Going further, we demonstrated in (Hohenstein et al., 2014) how to set up a hybrid cloud scenario. A hybrid cloud uses both a public and a private cloud. The motivation for hybrid clouds is often to keep critical or privacy data on private servers. One reason might be regulatory certifications or legal restrictions forcing one to store material that is legally relevant or subject to possible confiscation on premises. However, non-critical data could be routed to more efficient cloud offerings from external providers, which might be cheaper and offer better extensibility. The idea is to control the location of objects according to metadata.

The federation again occurs at container level: A logical container can be split across physical public and private cloud containers. Every logical container has to know its physical locations. To this end, we make the two cloud containers aware of each other by a PUT request with the payload of Figure 4, which has to be sent to the federation service of vision-tb-1. An analogous PUT request is implicitly sent to the second cloud, however, with an ”inverted” payload.

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

Figure 4: Federation payload.

This configuration contains information regarding the private and public cloud types, URLs, users, authorization information etc. Due to `private/public_cloud`, *vision-tb-1* will be the private cloud and *vision-tb-2* the public cloud. Such a specification is needed for any container (here logicalContainer) to act as a shard.

Clients are enabled to distribute data over the clouds and are provided with a unified view of the data that resides in both the private and public cloud. The hybrid cloud setup is completely transparent for the clients of a container, and the clienst might even not be aware where the data resides. Data CRUD operations are performed in a sharded way; The possibility to determine data confidentiality is given to the client by a metadata item that indicates data confidentiality. In order to store confidential data, one need to perform the request in Figure 5.

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

Figure 5: PUT request for storing confidential data.

PUT requests are handled as follows by a new ShardService in the CCS. The metadata of the object is checked for an item indicating confidentiality such as *confidential* : *true*|*false*. The approach benefits from the ability of the CCS to connect to multiple object storage nodes at the same time using the basic federation component of VISION Cloud. The sharding is performed in the CCS based on metadata values. According to the metadata of the container, the connection information is determined for both clouds. The ShardService decides to store *newObject* in the *private* cloud.

Every GET request to a federated container is sent to all clouds participating in the federation unless confidentiality is part of the query. The results of requests are combined, and the result is sent back to the client.

Each federated cloud can be an access point to the federation, i.e., can accept requests. Hence, there is no additional interface to which object creation operations and queries need to be submitted. A request can be sent to any shard in any cloud. Access to the container can be via the public and private cloud store, accesses are delegated to the right location.

The basic federation service of VISION Cloud (cf. 2.5.1) is not used for the hybrid cloud setup. Instead, the service has been technically implemented in the CCS that provides the content-centric storage functions. 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.

To sum up, the development of the VISION Cloud

project mainly provides important mechanisms that can be used as a base for federations:

1. *Metadata concept.* All objects are allowed to contain user-definable metadata entries. Those key-value pairs can be used in several ways to query for objects inside the cloud storage system. A schema can be employed for these metadata entries to require the existence of certain metadata fields and hence to enforce a certain metadata schema.
2. *Adapters for storage clouds.* VISION Cloud includes an additional layer that abstracts from the underlying storage and thus makes it possible to integrate cloud storage systems, e.g a blob storage service of some larger public cloud offering.

3 MULTI-CRITERIA STORAGE APPROACH

In the previous Section 2, we have considered an onboarding and a hybrid cloud setup where the location of objects is determined according to metadata. Being able to work with several cloud storage systems in a sharded manner offers further opportunities which are explained in this section.

3.1 The Approach and Concept

The goal is to extend the existing VISION Cloud federation approach to offer a more flexible solution for a so named multi-criteria storage. The term multi-criteria refers to the ability of the approach to organize the storage in shards based on multiple criteria at the same time.

This work uses the concept of federation (Vernik et al., 2013) to define a heterogeneous cloud setup involving different cloud storage systems with various properties and benefits. The metadata processing facilities are used to let clients specify storage criteria to be satisfied. The metadata processing takes places in the Content Centric Service component of VISION Cloud (Jaeger et al., 2012), using the management models and their associations.

Let us assume several cloud storage systems (CSs) named CS_1, CS_2, CS_3 etc. Each CS_i possesses specific properties and advantages with regard to privacy, access speed, availability SLAs and other characteristics following a resource model (cf. 3.2.2).

The definition of CS properties requires an administrative PUT request to the federation admin service of a VISION Cloud installation with a payload that describes the capabilities of any added CS_i. These properties are for the whole CSs, not for a specific container or object. The payload is then automatically distributed to all the members of the federation. At this point, the difference to the previously described VISION Cloud federation approaches becomes obvious: instead of federating containers, the multi-criteria approach spans across several containers globally. The extension of the federation concept intends to use different cloud storages in different provisioning modes. Referencing a classical example, data with higher demands in terms of privacy should be routed to locally hosted cloud storage systems, while data subject to less critical could be stored in a public cloud offering. As another example, cloud storage providers have special features such as reduced resilience. Storage with reduced redundancy can be used to store data that is temporarily required, in such a cheaper cloud storage offering.

Technically, the intended goal is to establish metadata conventions that provide rules for processing object or containers metadata in order to take routing decisions to the according storage. Thus, CCI operations obtain a higher level of abstraction since hiding the various underlying storage systems. The behavior of existing CCI operations is affected as follows:

- **To Create a New Container.** A PUT request can be sent to the CCI of any CS_i specifying the desired properties according to a criteria model (cf. 3.2.1): The receiving CS_i decides where to create the container in order to satisfy the specified properties. Usually, there will be one CS_i, however, several CSs might be suitable and chosen for storing data, e.g., for sharding data, load balancing, or achieving higher availability by redundancy. Any CS_i is able to handle requests by involving other CSs, if necessary. The container keeps metadata about its properties and a mapping to its list of further containers in other storage systems.
- **To Create a New Object (in a Container).** Again, a PUT request can be sent to the CCI of any CS_i. We decided to allow for specifying criteria not only for a container but also an object. Thus, the container criteria lead to a default storage location for all its objects, which can be overridden on a per-object basis. This principle supports use cases where a ”logical” container should store images, however, some of them are confidential (e.g., non-anonymous press images) while others are non-confidential (e.g., anonymous versions of the same press images), leading to different CS_i. Some objects are rarely accessed while others are used frequently. Some of them are relevant on the long term while other images could be easily recovered (i.e., thumbnails of original

images) and thus do not require high reliability. The CS_i takes the same decisions as above to determine the appropriate CS(s).

- **To Retrieve a Particular Object in a Container.** If a client wants to get an object, each CS must be able to retrieve the object even it is not available on its CS. In that case, the object's metadata is not available with such a request. Therefore, every CS_i must determine the possible object locations from the globally propagated properties. Anyway, querying the CS's is performed in parallel.

Please note VISION cloud deployment sets up a federation admin service/interface for each cloud storage. In general, all different cloud storage nodes provide the same service interfaces.

3.2 Requirements and Resource Model

One of the main benefits of a federation approach is that data can be placed optimally on different storage systems according to user and system specified functional and non-functional requirements. In order to achieve this goal, properly capturing the application requirements is needed as well as an accurate description of the underlying physical resources. Therefore the VISION Cloud management models (Gogouvitis et al., 2012), requirements and resource model, come into play in federation scenarios.

3.2.1 Requirements Model

In order for an application to run effectively over a cloud infrastructure, the customer should be able to specify requirements, which will be used by the infrastructure to drive the data access operations of the application. To this end, a requirements model is necessary to capture the requirements emerging from application attributes modeling and the ones deriving directly from the user needs. In addition, such a model defines structures to describe lower level requirements for the service offerings of the cloud as well as resource requirements that are used for the resource provisioning. More specifically the following models have been developed:

- *User Requirements Model.* This model captures the user requirements in a formalized way. They can be high level requirements characterizing the application data or describing the needed storage service and can be translated to low-level storage characteristics. Examples of parameters described are metrics like number of users, durability and availability requirements.
- *Resource Requirements Model.* This model aims at specifying the resource requirements for the operation of the cloud service. This structure will be utilized during resource provisioning and will keep the desired resources for meeting the constraints specified by the application.

Criteria that can be specified for creating containers or objects (and which determines the placement in a storage system) are for instance:

- Confidentiality (private vs. public cloud, high-secure data center)
- Storage cost (at certain levels)
- Reliability
- Access speed (fast disk access and low latency due to geo-location)
- Load balancing or replication properties

By using these two types of models, a user of a cloud storage system is able to define his strategy regarding placement of data and criteria-based provisioning. There are important issues, being combinable and ranked with a percentage.

3.2.2 Resource Model

The purpose of the resource model is to describe in a uniform way different features of storage systems that make up a federation. The resource models consists among others of properties such as:

- Public/private cloud
- Cloud provider and type (e.g., Amazon S3 Blob Store)
- Price scheme for storage, basically per GB/month, number of transactions, data transfer etc.
- Redundancy factor
- Disk access properties (e.g., SSD)
- Latency of data center for locations

4 TECHNICAL ASPECTS OF MAPPING APPROACH

Indeed, the usage-related criteria must be mapped to the physical storage properties by relating the concepts of the requirements and resource models. Then, it is possible to map high-level requirements, described as metadata, to low level resource requirements and to finally find storage systems that can fulfill the specified user criteria.

As the general approach of VISION Cloud was to implement the CDMI, a user of the system should perform the configuration of the system using the same REST/JSON-based approach as CDMI does. As

such, the already federation service provides some configuration calls that are extended by a PUT operation to let an administrator submit configurations. All users of the interface can use generic REST clients as well as implement their own front end using these REST calls.

Inside the federation service, one important task is to map the criteria specified by users to the technical parameters of storage systems in such a way that a storage system fitting best to the criteria will be found. To this end, we implemented an appropriate mapping approach. The basic idea is as follows:

- Each storage system is described by certain categories according to the resource model: public/private, a redundancy factor, its location, access speed, latency etc. (cf. Figure 6). The payload of Figure 4 obtains the properties of every newly added CS.
- If a user request to create a container or object with associated metadata requirements arrives at the CCS, the CCS asks the federation service for the federation information. The request contains metadata according to the requirements model – independently of the physical resource model of CSs. The federation returns the referring storage location(s) accordingly. All the metadata is part of a request, similar to Figure 5.

Storage System	type	public	provider	redundancy factor	access speed	location
CS_1	SSD disk	no	Samsung	1	1ms	local
CS_2	Vision store	no	Vision	2	10ms	local
CS_3	Blob store	yes	Amazon	3	5ms	Amsterdam
CS_4	Table store	yes	Azure	3	20ms	Dublin
CS_5	Blob store	yes	Azure	3	8ms	Dublin

Figure 6: Table RESOURCE – storage system properties.

The basis for recommending appropriate Storage Systems (CSs) for given user requirements is first a list of storage systems (cf. Figure 6) with their properties. The task of producing a recommendation is often formulated in knowledge-based systems as a tuple (R,E). R corresponds to the set of user requirements and E is the set of elements that form the knowledge base. In our case, the elements E are the features of the resource model, i.e., entries in Table 6.

The solution for a task (R,E) is a set S $\subseteq$ E that has to satisfy the following condition:

$\forall\, e_i \in S : e_i \in \sigma_{(R)}(E)$

$\sigma_{(R)}(E)$ is a selection on those elements in E that satisfy R. As an example, if the user requirements are defined by the concrete set R = $\{$ r_1 : access speed $\leq$ 15; r_2 : public = yes $\}$, then obviously only CS_5 satisfies the requirement R.

We now propose a procedure that allows recommending appropriate storage systems. To determine a ranking, we apply a schema that forms the basis for a Multi-Attribute Utility Theory (MAUT). Using this theory, resource requirements can be assessed and ranked according to the dimensions of a particular interest. Dimensions of interest are in our case user requirements such as reliability or cost.

	Confidentiality	Reliability	Cost Savings	...
public=yes	1	5	8	
public=no	9	2	2	
redundancy factor = 1		1	9	
redundancy factor = 2		3	5	
redundancy factor > 2		9	1	
provider=Amazon		7	5	
provider=Azure		7	4	
provider=Samsung		5	3	
Provider=Vision		8	5	
access speed >= 30ms			8	
access speed < 30ms			6	
access speed < 20ms			4	
access speed < 10ms			1	
...				

Figure 7: Table MAP – mapping table.

To set up a MAUT schema, the properties of the storage systems are assigned to the dimensions of interest in a first step ending up in a matrix. Each entry of the matrix contains a value that defines the relevance for the related dimension by associating a certain weight. The larger the value is, the more contributes the property to the dimension of interest. Figure 7 presents a sample mapping matrix. The matrix should be understood as follows. The first line ('public=yes') specifies that a public infrastructure as a resource requirement contributes to

- confidentiality (second column) with a weight of 1, thus being quite low;
- reliability with a medium weight of 5;
- cost savings with a high weight of 8 etc.

Similarly, 'private=no' contributes to confidentiality with a weight of 9, to reliability with a weight of 2, to cost savings with a weight of 2 etc.

The redundancy factor has impact on the reliability (increasing with the factor), to the costs (decreasingly) etc., but not on confidentiality.

A user U then formulates his requirements, which represent his preferences in a request. Finding adequate storage systems implies that these requirements have to be satisfied. Moreover, a user can rank each of its features with a percentage, i.e., a user has the possibility to define a WEIGHT(U,d) for each dimension d of interest in a request. For instance, a user U can specify a weight of 50% for the dimension "Con-

fidentiality" and weights of 30% for the dimension "Reliability" and 20% for the dimension "Cost Savings". The storage CS_i with the highest overall value is suited best to satisfy the demands.

In order to formalize the approach, we assume a function $MAP : Properties \text{ x } Values \text{ x } Dimensions \rightarrow Int$ that represents the mapping table. For instance, $MAP(public, y, Reliablity)$ refers to the value 5 in the first line in Figure 7.

Another function $RESOURCE : StoragesystemsxProperties \rightarrow Values$ corresponds to Figure 6.

The following formula then computes the relevance of a storage system in a weighted manner using those functions:

$Relevance(U, CS_i) = \Sigma_{d \in Dimensions} (\Sigma_{p \in Properties} MAP(p, RESOURCE(CS_i, p), d) \times WEIGHT(U, d))$

Using this formula, we can calculate for each CS_i a value for each dimension and obtain the resulting table in Figure 8 for our example. That is, CS_2 having a value of 10.4 is best suited for the given set of requirements whilst CS_3 and CS_5 with a value of 7.7 are the worst options.

Storage System	Confidentiality (50%)	Reliablity (20%)	Cost Savings (30%)	Overall Result
CS_1	5	2 + 1 + 5	2 + 9 + 1	9.7
CS_2	5	2 + 3 + 8	2 + 5 + 4	10.4
CS_3	1	5 + 9 + 7	8 + 1 + 1	7.7
CS_4	1	5 + 9 + 7	8 + 1 + 6	9.2
CS_5	1	5 + 9 + 7	8 + 1 + 1	7.7

Figure 8: Sample calculation.

Due to the logic applied to the generation of a recommendation, it might happen that a given set of requirements leads to an empty set of recommendations: $\sigma_{(R)}(E) = \emptyset$. In such a case, it is not very helpful to inform the user only about such a conflict, but also to give him a hint about what requirement should be relaxed in order to obtain a recommendation.

A set of conflicts is a set SoC $\subseteq$ R such that $\sigma_{SoC}(E) \neq \emptyset$. SoC is maximal in the sense that no other conflict SoC' fulfilling $\sigma_{SoC'}(E) \neq \emptyset \wedge$ SoC' $\supset$ SoC exists. A set of conflicts SoC refers to a certain elements e $\in$ E under consideration of a set R of requirements. SoC gives a concrete hint about what condition to relax.

5 RELATED WORK

Even if cloud federation is a research topic, the basic concepts and architectures of data storage federations have already been discussed many years ago within the area of federated database management systems (Sheth and Larson, 1990). 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". (Sheth and Larson, 1990) describes a five-layer reference architecture for federated database systems. According to the definition, the federated database layer sits on top of the contributing component database systems.

One possible characterization of federated systems can be done according to the dimensions of distribution, heterogeneity, and autonomy. One can also differentiate between tightly coupled systems (where administrators create and maintain a federation in advance) and loosely coupled systems (where users create and maintain a federation on the fly).

Based upon Google App Engine, Bunch et al. (Bunch et al., 2010) present a unified API to several data stores of different open source distributed database technologies. Such a unified API represents a fundamental building block for working with cloud storage as well as on-premises NoSQL database servers. However, the implementation provides access only to a single storage system at a time. Hence compared to our CCS solution, the main focus is on portability and not on a federated access.

Redundant Array of Cloud Storage (RACS) is a cloud storage system proposed by Abu Libdeh et al. (Abu-Libdeh et al., 2010). RACS can be seen as a proxy tier on top of several cloud storage providers, and offers a concurrent use of different storage providers or systems. Adapters for three different storage interfaces are discussed in the paper, however, the approach can be expanded to further storage interfaces. Using erasure coding and distribution, the contents of a single PUT request are split into parts and distributed over the participating storage providers similar to a RAID system. Any operation has to wait until the slowest provider has completed the request. While this approach splits data across storage systems, our approach routes a PUT request to the best suited storage system.

Brantner et al. (Brantner et al., 2008) build a database system on top of Amazon's S3 cloud storage with the intention to include support for multiple cloud storage providers in the future. In fact, Amazon S3 is also one of storage layer options that VISION supports.

There is a lot of ongoing work in the area of multi-cloud APIs and libraries. Their goal is also to enable a unified access to multiple different cloud storage systems. Among them, Apache Libcloud (Libcloud,

2017), Smestorage (SmeStorage, 2017) and Deltacloud (Deltacloud, 2017) should be mentioned. They provide unified access to different storage systems, and protect the user from API changes. However, only basic CRUD methods are supported, mostly lacking of query functionality. Moreover, they have administration features such as stopping and running storage instances.

Further notable approaches can be found in the area of Content Delivery Networks (CDN). A content delivery network consists of a network of servers around the world which maintain copies of the same, merely static data. When a user accesses the storage, the CDN infrastructure delivers the website from the closest servers. However, Broberg et al. (Broberg et al., 2009) state that storage providers have emerged recently as a genuine alternative to CDNs. Moreover, they propose a system called Meta CDN that makes use of several cloud storage providers. Hence, it is not really a storage system. As CDNs in general, Meta CDN mostly focuses on read performance and neglects write performance, too. Anyway, the system provides cheaper solution by using cloud storage.

There are also a couple of hybrid cloud solutions in the literature. Most of them focus on transferring data from private to public, not providing a unified view to hybrid storages. For instance, Nasuni (Nasuni, 2017) is a form of network attached storage, which moves the user's on-premise data to a cloud storage provider. Hence, this hybrid cloud approach gather and encrypt the data from on-premise storage, afterwards sending the encrypted data to a public cloud at either Microsoft Azure or Amazon Web Services. Furthermore, a user has the option to distribute data over multiple stores. Compared to our multi-level sharding approach, Nasuni is a migration approach that eventually moved data to the public cloud.

Another example is Nimbula (Nimbula, 2017), which 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 (CloudSwitch, 2017) has also developed a hybrid cloud that allows an application to migrate its data to a public cloud.

A hybrid cloud option has been developed by Nirvanix (Nirvanix, 2017), however, based upon proprietary Nirvanix products and thus limiting usage due to a danger of vendor lock. Nirvanix offers a private cloud on premises to their customers, and enables data transfer to the public Nirvanix Cloud Storage Network. Other public cloud platforms than Nirvanix are not supported. Due to our adapter approach, VISION is not limited to a specific public cloud service.

As a general observation that can be made, most commercial federated and hybrid cloud storage solutions do provide a range of offerings to satisfy various customers demands, but pose the risk of a vendor lock-in due to the use of own infrastructure.

The MetaStorage (Bermbach et al., 2011) system seems to be most comparable to our approach since it is a federated cloud storage system that is able to integrate different cloud storage providers. MetaStorage uses a distributed hash table service to replicate data over diverse storage services by providing a unified view between the participating storage services or nodes.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented a multi-level federation approach for storing large objects like videos. The approach is based upon the metadata idea of the VISION Cloud project. Our approach provides a uniform interface for accessing data. The key idea is to use metadata for controlling data placement behavior at a higher semantic level by specifying requirements instead of physical properties. As a technical basis, we benefit from the VISION Cloud software stack (VISION-Cloud, 2011) where such a metadata concept is an integral part for handling storage entities. We show in detail how well-suited VISION Cloud and its storage system architecture is to support new scenarios such as using several storage technologies with different properties for different purposes in parallel, e.g., handling confidential and non-confidential data, the first kept in an on-premise data store, the later stored in a public cloud.

In this respect, the overall approach allows for adding various further sharding strategies, such as region-based, load balancing, or storage space balancing, redundancy level control etc.

In our future work, we are evaluating the approach. Furthermore, several points have not been tackled so far and are subject to future work. For instance, changing or adding properties of a storage system might lead to a redistribution, maybe taking benefit from on-board federation/migration. Similarly, adding a new storage system node also affects the distribution to storage systems. And finally, we think of extending the approach to a self-learning system that uses information about the user's satisfaction to adopt the mapping table between users' requirements and the properties of the storage system.

ACKNOWLEDGEMENTS

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

Abu-Libdeh, H., Princehouse, L., and Weatherspoon, H. (2010). Racs: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 229–240, New York, NY, USA. ACM.

Bermbach, D., Klems, M., Tai, S., and Menzel, M. (2011). Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 452–459, Washington, DC, USA. IEEE Computer Society.

Brantner, M., Florescu, D., Graf, D., Kossmann, D., and Kraska, T. (2008). Building a database on s3. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, page 251.

Broberg, J., Buyya, R., and Tari, Z. (2009). Service-oriented computing — icsoc 2008 workshops. chapter Creating a 'Cloud Storage' Mashup for High Performance, Low Cost Content Delivery, pages 178–183. Springer-Verlag, Berlin, Heidelberg.

Bunch, C. et al. (2010). An evaluation of distributed datastores using the appscale cloud platform. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 305–312, Washington, DC, USA. IEEE Computer Society.

CDMI (2010). Cloud data management interface version 1.0. At: http://snia.cloudfour.com/sites/default/files /CDMI_SNIA_Architecture_v1.0.pdf. [retrieved: March, 2017].

CloudSwitch (2017). Cloudswitch. At: http://www.cloudswitch.com. [retrieved: March, 2017].

Deltacloud (2017). Deltacloud. At: http://deltacloud.apache.org/. [retrieved: March, 2017].

Fielding, R. T. and Taylor, R. N. (2002). Principled design of the modern web architecture. *ACM Transactions on Internet Technologies*, 2(2):115–150.

Fox, A. et al. (2009). Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28.

Gogouvitis, S. V., Katsaros, G., Kyriazis, D., Voulodimos, A., Talyansky, R., and Varvarigou, T. (2012). Retrieving, storing, correlating and distributing information for cloud management. In Vanmechelen, K., Altmann, J., and Rana, O. F., editors, *Economics of Grids, Clouds, Systems, and Services*, volume 7714 of *Lecture Notes in Computer Science*, pages 114–124.

Hohenstein, U., Jaeger, M., Dippl, S., Bahar, E., Vernik, G., and Kolodner, E. (2014). An approach for hybrid clouds using vision cloud federation. In *5th Int. Conf. on Cloud Computing, GRIDs, and Virtualization (Cloud Computing), Venice 2014*, pages 100–107.

Jaeger, M. C., Messina, A., Lorenz, M., Gogouvitis, S. V., Kyriazis, D., Kolodner, E. K., Suk, X., and Bahar, E. (2012). Cloud-based content centric storage for large systems. In *Federated Conference on Computer Science and Information Systems - FedCSIS 2012, Wroclaw, Poland, 9-12 September 2012, Proceedings*, pages 987–994.

Kolodner, E. et al. (2011). A cloud environment for data-intensive storage services. In *CloudCom*, pages 357–366.

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

Libcloud (2017). Apache libcloud: a unified interface to the cloud. At: http://libcloud.apache.org/. [retrieved: March, 2017].

Mell, P. and Grance, T. (2011). The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7.

Nasuni (2017). Nasuni. At: http://www.nasuni.com/. [retrieved: March, 2017].

Nimbula (2017). Nimbula. At: http://en.wikipedia.org/wiki/Nimbula. [retrieved: March, 2017].

Nirvanix (2017). Nirvanix. At: http://www.nirvanix.com/products-services/cloudcomplete-hybrid-cloud-storage/index.aspx. [retrieved: March, 2017].

NoSQL (2017). Nosql databases. At: http://nosql-database.org. [retrieved: March, 2017].

Sandalage, P. and Fowler, M. (2012). Nosql distilled: a brief guide to the emerging world of polyglot persistence. *Pearson Education*.

Sheth, A. and Larson, J. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, (22 (3):183–236.

SmeStorage (2017). Smestorage. At: https://code.google.com/p/smestorage/. [retrieved: March, 2017].

Vernik, G. et al. (2013). Data on-boarding in federated storage clouds. In *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, CLOUD '13, pages 244–251, Washington, DC, USA. IEEE Computer Society.

VISION-Cloud (2011). 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: March, 2017].

VISION-Cloud (2012). 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: March, 2017].