Determining Appropriate Large Object Stores with a Multi-criteria Approach

Uwe Hohenstein[1(✉)], Spyridon V. Gogouvitis[2(✉)], and Michael C. Jaeger[1(✉)]

[1] Corporate Technology, Siemens AG, Otto-Hahn-Ring 6, 81730 Munich, Germany
{uwe.hohenstein, michael.c.jaeger}@siemens.com
[2] Mobility Division, Siemens AG, Otto-Hahn-Ring 6, 81730 Munich, Germany
gogouvitis@siemens.com

Abstract. The area of storage solutions is becoming more and more heterogeneous. Even in the case of relational databases, there are several offerings, which differ from vendor to vendor and are offered for different deployments like on-premises or in the Cloud, as Platform-as-a-Service (PaaS) or as a special Virtual Machine on the Infrastructure-as-a-Service (IaaS) level. Beyond traditional relational databases, the NoSQL idea has gained a lot of attraction. Indeed, there are various services and products available from several providers. Each storage solution has virtues of its own even within the same product category for certain aspects. For example, some systems are offered as cloud services and pursue a pay-as-you-go principle without upfront investments or license costs. Others can be installed on premises, thus achieving higher privacy and security. Some store redundantly to achieve high reliability for higher costs. This paper suggests a multi-criteria approach for finding appropriate storage for *large* objects. Large objects might be, for instance, images of virtual machines, high resolution analysis images, or consumer videos. Multi-criteria means that individual storage requirements can be attached to objects and containers having the overall goal in mind to relieve applications from the burden to find corresponding appropriate storage systems. For efficient storage and retrieval, a metadata-based approach is presented that relies on an association with storage objects and containers. The heterogeneity of involved systems and their interfaces is handled by a federation approach that allows for transparent usage of several storages in parallel. All together applications benefit from the specific advantages of particular storage solutions for specific problems. In particular, the paper presents the required extensions for an object storage developed by the VISION Cloud project.

Keywords: Cloud storage · Federation · Multi-criteria · NoSQL

1 Introduction

According to the National Institute of Standards and Technology (NIST) [18] 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,

J. Filipe et al. (Eds.): DATA 2017, CCIS 814, pp. 128–153, 2018.
https://doi.org/10.1007/978-3-319-94809-6_7

applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. One important resource that can be provisioned by a Cloud is obviously storage. In particular, Cloud storage offers all the benefits that are described by [11] for Cloud computing in general. A user can ask for virtually unlimited storage space, provisioned in a couple of minutes, thereby paying only for the provisioned storage. This is known as the pay-as-you-go principle and implies that no long-term commitments and no upfront costs apply to usage. Moreover, users do not have to take care about software licenses (e.g., as for database products) since these are already included in the usage fees.

Cloud storage solutions usually include Blob-stores for large unstructured objects and NoSQL databases [22], the latter trying to establish an antipole to traditional relational (i.e., SQL) databases. In fact, the term "NoSQL" stands more moderately for "not only SQL". NoSQL databases abdicate the well-known transactional ACID properties of SQL databases in favor of highly distributing data across several computer nodes, thus benefiting from parallelism. This perfectly fits to cloud computing environments. Furthermore, NoSQL databases offer more flexibility by not relying on a pre-defined, fixed table schema. Schema-less data can be stored in flexible NoSQL structures, i.e., the kind of data and its structure does not have to be known in advance.

Despite new arising storage technologies, well-known relational databases and providers are also offered by cloud providers. Consequently, relational databases are also a kind of cloud storage in a broader sense.

In particular, NoSQL is again a placeholder for a lot of different approaches and products, also covering "elder" technologies such as object-oriented or XML databases. Consequently, an increasing heterogeneity of storage technologies is currently arising, also keeping in mind the product or vendor specific and variants within a certain category.

In general, there is no best storage technology and no best product for a technology. Hence, the goal is to find a storage solution that best fits to a certain application. In case of complex applications with different storage demands, even a combination of different storage solutions makes sense. Sandalage and Fowler [23] shaped the term "polyglot persistence" for such a scenario. The idea is to use the most appropriate storage technology for each specific problem. "Appropriate" might cover several aspects such as the functionality, e.g., a full power of SQL queries, but also cost aspects, latency, reliability, or security issues. For example, costs can be decreased by using slow and cheap storage whenever possible instead of fast but expensive storage. Costs can also be reduced by using a public cloud offering. However, the various and complex price schemes and underlying factors such as price/GB, data transfer or number of transactions have to be understood. However, a public cloud storage might not be the best solution for storing highly confidential data due to possible security risks or compliance with legal regulations. A hybrid cloud, i.e., combining storage in a private cloud on premises for keeping confidential data with a public cloud for non-confidential data, might be appropriate.

This paper is an extension of [14], where we suggested a multi-criteria, federation approach to cover those scenarios. The overall goal is to find an optimal data placement solution for an overall benefit. To this end, we extend our work for a cloud storage developed by the European funded VISION Cloud project [16, 17]. In contrast to

conventional solutions, the overall goal of VISION Cloud is to develop next generation technologies for storing and efficiently retrieving *large* objects such as virtual machine images, high resolution imaging analysis scans, or videos.

In the following, Sect. 2 introduces the VISION Cloud software as far as it is relevant for this work, particularly, the storage architecture and the major storage interfaces. The VISION Cloud approach is called content-centric and follows the CDMI proposal [7]: There is a clear distinction between objects and containers holding these objects. The fundamental concept for retrieval is a first-class support for metadata for objects and containers. We combine the metadata concept with a federation approach [24] to cover the multi-criteria scenarios.

Therefore, we explain in Sect. 3 the original VISION Cloud federation approach that has been extended to tackle the needs of a multi-criteria storage solution. The basic VISION Cloud federation support has been used to implement two simple scenarios, which are also discussed in Sect. 3.

Afterwards, Sect. 4 introduces the new extended federation multi-criteria approach. A more flexible federation approach provides a unified and location-independent access interface, i.e., transparency for data sources, while leaving the federation participants autonomous. The presented federation approach allows taking benefit from the advantages of various storage solutions, private, on-premises and public clouds, access speed, best price offerings etc. In addition to the previous work [14], we detailed the approach and explain in Sects. 4.9 and 4.10 some further ideas about an advanced monitoring and recommendation system that better supports users. Moreover, the approach is extended to incorporate feedback from users in an appropriate manner.

In Sect. 5, we discuss related work and underline the novelty of the approach. Finally, Sect. 6 concludes the paper and mentions some future work.

2 VISION Cloud Overview

2.1 Motivation

VISION Cloud [16] was a project co-funded by the European Union for the development of cloud storage solutions with a specific focus on large objects such as videos, high resolutions diagnostics images, and virtual machine images. The overall VISION Cloud goal was to develop a storage system whereupon several domain applications are enabled to raise innovations. VISION Cloud should particularly support domains that have an increasing need for a large capacity and expect to store a vast number of large objects while also being able to retrieve stored content efficiently. Thus, typical domains targeted by VISION Cloud were health care, broadcasting and media, and IT application management, which have advanced needs for suitable storage systems due to the increasing storage capacity and large storage consumption per object and handling of very large objects. Indeed, sharing of video messages in the telecommunication domain turns into a trend since the market share of high-resolution camera equipped handsets grows [27]. Similarly, Ultra High-Definition and 4 K resolution content becomes a standard in the media domain.

In conventional approaches, large objects are stored in files and organized in a hierarchical structure. Hence, the hierarchical structure is the only mean to find a particular item by means of navigating through the hierarchy until an object is found. In fact, the hierarchy has to be set up manually and in advance without knowing all the search criteria. Hence, such a hierarchy is prone to become outdated soon or later. If the hierarchy is not understood correctly, objects might also be wrongly placed. This all leads to difficulties in finding objects in an efficient manner, especially in case of larger volumes of data. In addition, the problem arises how to maintain hierarchy changes in a distributed environment.

In order to overcome those problems, VISION Cloud pursues a metadata-centric approach and explicitly represents the type and format of the stored objects. The content-centric approach relieves a user from establishing hierarchies in order to organize a high number of storage objects. Moreover, VISION Cloud puts a lot of emphasis on flexible search options with intuitive categories for ever increasing amounts of data and acceptable access performance.

2.2 Metadata Concept

VISION Cloud relies on metadata to store and retrieve objects and the ability to perform autonomous actions on the storage node based on metadata [16]. Any content is accessed based upon using metadata. Metadata is often already available such as the author or the title of a video, a description, and ratings of users who have used the content. More technical metadata is the file format, the date of upload, and the image compression algorithm, to give some examples. Hence, metadata often exists and be easily derived when being stored, as one of the demonstrators of VISION Cloud has shown [15]. Further metadata can also be attached manually in order to classify or characterize the content. A specific VISION Cloud concept, called *storlets* [17], enhances the possibilities to derive metadata significantly. Storlets are similar to triggers in relational databases; they contain executable code and are triggered by certain events. Hence, storlets are an excellent mean to analyze and process metadata from uploaded storage objects, and to attach derived metadata to objects. For example, the storlet approach can be applied to run speech-to-text analyzers on video content in order to extract the audio track as metadata and to store the resulting text as part of the video object.

Metadata acts as search terms and improves the ability to navigate across a large number of video objects and to efficiently retrieve objects. It is important to note that metadata can be attached to objects and containers, the latter keeping several objects. Indeed, container metadata contributes to retrieving the objects inside.

To keep metadata, VISION Cloud uses a key-value objects tree in a specific storage being separated from the proper storage for the large objects. Both the storage objects and their metadata are efficiently kept in synchronization. This is an important feature and technical challenge for the internal distributed, node-based architecture targeting at horizontal scaling.

2.3 Content Centric Interface

All the applications of VISION Cloud use an interface named *Content Centric Interface* (CCI). The CCI conforms to the Cloud Data Management Interface specification [7] because CDMI provides a standard for interfaces to general object storage systems. In CDMI, storage objects are basically organized in *containers*. Containers are similar to the concept of buckets in other storage solutions such as Amazon S3. Moreover, CDMI standardizes the access to and storage of objects in a cloud by offering the typical CRUD (Create, Retrieve, Update, Delete) operations in a REST style [10] using HTTP(S).

The VISION Cloud CCI extends CDMI as, e.g., the important concept of the previously mentioned storlet is not covered by the standard. Furthermore, the major goal of VISION Cloud is not only the organization of storage objects in containers, VISION Cloud also stresses on efficiently handling and querying objects in general by using the fundamental metadata concept.

In order to create a new container for a specific tenant `MyTenant` in in the CCI, a request `PUT /CCS/MyTenant/MyContainer` has to be issued. The payload of the request contains the metadata to be attached with that container. Figure 1 shows a complete request for creating a new container:

```
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" } }
```

Fig. 1. HTTP PUT request.

Similarly, a CCI request `PUT /CCS/MyTenant/MyContainer/MyObject` stores an object `MyObject` in the container `MyObject`. Again, the payload contains the metadata describing an object. To distinguish a container from an object, the type of data for this request is indicated by `cdmi-container` in the HTTP Content-Type header.

2.4 Vision Cloud Architectural Layering

Since the general VISION Cloud architecture was presented in detail in [16], we here concentrate on the major concepts for storage and the metadata handling. The VISION Cloud architecture is layered. The fundamental *object storage* layer provides the storage and management of metadata. This layer also handles the replication of storage objects with their metadata across the storage nodes.

Every node has a so-called *Content Centric Service* (CCS) deployed, which sits on top of the object storage layer. The CCS implements the CCI (cf. Subsect. 2.3) and

provides higher-level services such as relationships between storage objects, thus enhancing the CDMI standard with additional powerful operations for a rich metadata handling. As a popular use case, having a reference of an object, an application is enabled to retrieve similar objects in the storage. For instance, the provider of some content is able to store media content in a way that is optimized for high definition displays as found in smart phones or hand-held devices. Using the CCS, applications can query for objects that are similar to the object already known. This is pretty useful if different media encoding types for different end devices come into play. The CCS determines the encoding relations by means of key-value metadata. Indeed, similarity is a metadata feature that was especially introduced to support relations between objects as metadata. Internally, the CCS decodes the relation-based query issued by an application into an internal metadata format, which does not provide the higher relation concept. A corresponding query is sent to the underlying object storage.

At the top of the layered architecture, the real applications use the CCS to access the storage.

The software of the VISION cloud storage was developed to run on common appliances. In a VISION Cloud storage system, a huge number of parallel nodes can be deployed in a data center to achieve a high scalability and load balancing. Thereupon, applications are enabled to access several nodes as far as they possess a CCS and its underlying object storage stack. The distribution of data across several storage nodes remains transparent to the application.

The CCS applies an adapter concept to allow for an integration of different storage servers. Implementing such an adapter enables one to integrate other storage implementations within the CCS: As a prerequisite, any adapter has to take care of handling metadata besides the real object storage. That is, the CCS can work with other storage systems as well if they are capable of storing metadata attached to objects and containers. The open source CouchDB document database was integrated as one candidate into the CCS as part of the VISION Cloud project in order to prove applicability.

The CCS connects to a storage server's IP and port number by means of a specific storage adapter. Behind the IP, there might be a single storage server. But it is also possible that the IP address points to a load balancer with a cluster implementation behind. From a technical point of view, there is no need for the object storage and the CCS to reside on the same machine or node. The request handling can be easily deployed on the storage node as a module since it does not need to support sessions. Applications are allowed to bypass the CCS, thus accessing the object storage directly. In that case, it is however not possible to use the advanced CCS metadata handling capabilities. Furthermore, the CCS is capable of avoiding node management functionality and keeping track of the current status of object nodes. This all supports a general scale-out of VISION Cloud storage nodes.

The basic setup of VISION Cloud nodes deploys a CCS on every storage node, which is a (potentially virtual) server running a VISION Cloud node software stack. Hence, response times are reduced since connections between different network nodes are avoided.

3 Federations in VISION Cloud

3.1 Fundamental Support for Federations in VISION Cloud

In general, a federation has the goal to yield an access interface so that a transparency of data sources is achieved while at the same time let the federation participants stay autonomous. A Cloud federation should thus abstract from different storage clouds of different provisioning modes. Clients are offered by the federation a unified view of storage and data services across several systems and providers.

In particular, a federation has to tackle heterogeneity of the involved data sources. There are several types of heterogeneity. At first, each cloud provider offers proprietary Application Programming Interfaces (APIs) and management concepts. In fact, industry specifications such as CDMI [7] are a step towards homogenization. Furthermore, another type of heterogeneity to be handled by the federation is that cloud storage providers apply heterogeneous data models.

The implementation of the CCS of VISION Cloud already provides features for handling heterogeneity by means of an adapter concept. Each adapter abstracts from an underlying heterogeneous source by offering a common CCS interface. An instance of a VISION Cloud CCS sits on top of each storage system. CCS adapters are already implemented for the proprietary VISION Cloud storage service and CouchDB. Further adapters can be implemented for other storage systems. Consequently, the CCS approach allows for integrating multiple storage system types of various cloud providers.

Each CCS uses an IP connection to communicate with a storage system. In particular, a CCS can be based upon a load balancer that itself possess several homogeneous storage servers beneath and works as a cluster solution thereby bridging the load balancer and the client. Hence, all scalability, elasticity, replication, duplication, and partitioning is delegated to the storage system. Indeed, current storage system types usually have a built-in cluster implementation already. For example, the open source project MongoDB possesses several strategies for deploying clusters of MongoDB instances. And CouchDB provides an elastic cluster implementation called BigCouch. These cloud systems are able to deal with thousands of servers located in many data centers around the world thereby serving millions of customers. Hence, the complexity of CCS is drastically reduced by not re-implementing such distribution, scalability, load balancing, and elasticity features.

Moreover, VISION Cloud offers some fundamental federation functionality for distributing requests across multiple storage nodes. Using the metadata approach, the CCS is able to route storage requests to different storage services similar to a load balancer. Although not really representing a load balancer, the role of the CCS in the software stack is useful for similar purposes. For example, the CCS can distribute requests over different storage nodes depending on quality characteristics, matching the provided capabilities of the underlying storage clouds.

Furthermore, VISION Cloud integrates fine granular access control lists (ACLs), which can be attached to tenants, containers, and objects.

These basic VISION Cloud federation features have been used in two major federation scenarios, which are presented in the following subsections.

3.2 Using the Federation Concept for On-Boarding

According to [11], vendor lock-in is the second obstacle among the top ten for growth in cloud computing. A specific on-boarding federation supports clients to become independent of a particular cloud storage provider: On-boarding means that data can be transferred between clouds without operational downtime [26]. Hence, data can be accessed while the migration migrated from one storage system to another is in progress.

In a first step, an administrator has to set up an on-boarding federation between the two involved storage systems, the source and the target. To this end, the administrator sends a "federation configuration" to a federation service. This configuration specifies the access information of the remote (i.e., the source) container including credentials. Figure 2 describes the payload of a typical federation configuration.

```
"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"
}
```

Fig. 2. Sample payload.

The federation service is deployed along with the CCS and offers typical operations to configure and handle federations by means of REST-ful services. A federation instance can be created with a PUT request passing an id in the Uniform Resource Identifier (URI) and the federation configuration in the body. A GET-request returns the federation progress or statistical data for a particular federation instance. DELETE removes a federation configuration. More details can be found in the VISION Cloud project deliverable [28].

Having configured a federation for the source and target systems, the transfer of objects of federated containers can be initiated. While data is being transferred to the target container in the background, clients can still request the contents of the container even if the transfer of that container has not yet finished: All objects of all containers are available. Containers that have not been completely transferred will be fetched from the remote source by using the federation information. The federation service possesses an on-boarding handler that intercepts GET-requests from the client and redirects them to the remote system for non-transferred containers. For transferring data, the information is used by the target container to access the remote server and to fetch data from the remote to the target server. A federation works on a container level. Moreover, the handler is responsible for scheduling the background jobs for transferring data.

3.3 Hybrid Cloud Federations

In another scenario, a federation is used to set up a hybrid cloud [13], consisting of a public and a private cloud. Hybrid clouds are useful if both critical or privacy data and non-critical data has to be stored. For example, regulatory certifications or legal restrictions forces one to store material that is legally relevant or subject to possible confiscation on premises, on private servers. Since private servers are more expensive, it is not reasonable to keep non-critical data on the same private server. Instead, more efficient cloud offerings from external providers, which are cheaper and offer better extensibility, should be used. The idea is to split a "logical" container across physical public and private cloud containers and to use metadata in order to control the location of objects. That is, a PUT request for storing objects can obtain some metadata that specifies whether the data is confidential or not.

As a prerequisite, every logical container has to know its physical locations. This can be done by means of an administrative PUT request with the payload of Fig. 3, which has to be sent to the federation service of vision-tb-1. Internally, the second cloud receives an "inverted" payload. Afterwards, the two cloud containers know the locations of each other.

```
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"
}
```

Fig. 3. Federation payload for configuration.

The configuration in Fig. 3 specifies the private and public cloud types, URLs, users, authorization information etc. The `private/public_cloud` specification determines that the cloud server `vision-tb-1` will contain the private container and `vision-tb-2` the public one. Every container requires such a specification if being split into a private and public part. Hence, the federation occurs at container level.

The distribution of data across a private and public part is transparent to clients. A client is completely unaware of the hybrid cloud setup and does not know where the data is kept. During the creation of objects, metadata is used to determine data confidentiality by a metadata item `confidential:"true"|"false„` for instance in the request presented in Fig. 4.

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

Fig. 4. PUT request for storing confidential data.

To enable a hybrid scenario, a ShardService had to be introduced as part of the CCS. The CCS handles the PUT request by checking the metadata of the object and delegates the request to the public or private cloud accordingly. Indeed, the CCS provides all the content-centric storage functions. The ShardService implements a reduced CDMI interface and has the key role in hybrid scenarios by intercepting each request to a CCS and to decide where to forward the request. Internally, the connection information is determined for both clouds by using the hybrid cloud setup. In case of `"confidential":"true"„` the ShardService decides to store `newObject` in the private cloud. Thereby, the ShardService uses the ability of the CCS to connect to multiple object storage nodes at the same time. A client is able to connect to any Cloud server and does not need to know the specific location of a container.

Similarly, clients obtain a unified view of the data, no matter whether located in the private or public. Hence, a GET request to a federated container is sent to all clouds participating in the federation unless confidentiality is part of the query and the server is thus known. The results of the individual requests for the private and public containers are collected, and the combined result is returned to the client.

Every operation can be sent to the public and private cloud store in the federation, i.e., each cloud can accept requests and is able to answer. There is no additional interface to which object creation operations and queries need to be submitted.

4 Multi-criteria Storage Federations

The potential of the metadata approach for federation has already been illustrated in Sects. 3.2 and 3.3. Indeed, metadata is able to keep information that is then used to determine the storage location. The approaches are enhanced in this section to achieve a better flexibility of specifying and selecting cloud storage systems in order to take full benefit from their particular capabilities.

4.1 Basic Idea

The overall approach provides means to let a user influence the storage of data. For example, putting higher demands on privacy lets objects be stored in locally hosted cloud storage systems, while less critical data is stored in public cloud offerings. Furthermore, the approach benefits from special features of cloud storage providers such as reduced resilience: It is reasonable to store data that is only temporarily required in some cheaper cloud storage with reduced redundancy. The pursued approach is a so-called multi-criteria storage. The term multi-criteria thereby underlines the ability of the approach to organize the storage based upon multiple user criteria. Relying on the VISION Cloud federation concept [26] to define a setup of heterogeneous clouds, it becomes possible to involve several cloud storage systems each possessing various properties and benefits. Moreover, the metadata processing features of VISION Cloud build an important component of the multi-criteria approach. In fact, the idea relies on applying metadata to let users ask for individual storage criteria for both containers and objects.

The VISION cloud deployment sets up a federation admin service (including an interface) for each cloud storage participating in the federation. Every cloud storage node offers the same service interfaces.

Let us assume several cloud storage systems (CSs) named CS1, CS2, CS3 etc. Each of these CSs should possess dedicated properties with respect to availability SLAs, privacy issues, and access speed. Indeed, the CS properties follow the resource model (cf. Sect. 4.2).

The definition of properties for a particular CS requires an administrative PUT request to the federation admin service of a VISION Cloud. In contrast to the federation scenarios in Section [3], a different payload has to describe the capabilities of every CS – for the whole CS, not for particular containers or objects. Each member of the federation receives the payload to become aware of the CS and its particular properties.

The federation principle again benefits from the metadata approach and let CCI operations thus work at a higher level of abstraction by hiding the underlying storage systems and their interfaces. The existing CCI operations behave as described in the following subsections.

The Content Centric Service (CCS) component of VISION Cloud [15] has to process the metadata with the additional task of satisfying those requirements.

4.1.1 Creation of a New Container

A user has to issue a PUT request that contains metadata according to a requirements model (cf. Sect. 4.3), thus specifying the desired properties. The PUT request can be sent to the CCI of any CSi. Every container maintains metadata about its own properties and the mapping to its list of further containers in other storage systems. The CSi analyzes the request and is able to decide where to create the container in order to fulfill the specified requirements at best. In some cases, several CSs might be chosen, e.g., to achieve higher availability by keeping data redundantly. It should be noted that other CSs are involved in satisfying the storage request, if necessary. Every CSi is capable to receive and answer requests, even if the container will finally be directed to a different CS.

4.1.2 Creation of a New Object

Again, creating a new object in a container requires a PUT request with user requirements to be sent to the CCI of any CSi. As a general design decision, storage criteria can be attached to both containers and objects. Thus, we are able to support advanced use cases where a per-object decision is useful. For example, a "logical" container might keep images, some of them are confidential (e.g., anonymous versions of the same press images), while others are not confidential (e.g., non-anonymous press images). Accordingly, different CSi are used for storage. Some images could be easily recovered (i.e., thumbnails of original images) and thus do not require high reliability, while other objects are relevant on the long term. Or some objects might be rarely accessed while others are used frequently. In any case, the CSi takes the decision to choose the appropriate CS(s). Indeed, the container criteria lead to a default storage location for all its objects, which can be overridden on a per-object basis.

4.1.3 Retrieval of Objects

If a user wants to retrieve a particular object in a container, each CS must be asked unless the criterion determines one or several CSs. Hence, every CSi must be able to determine the possible object locations by means of using the metadata. The query is performed in all the CSs in parallel.

4.2 Resource Model

Our major goal is to use a federation approach in order to place data optimally on different storage systems according to user-defined functional and non-functional requirements. In a first step, the physical resources of the underlying storage solutions need to be precisely described. To this end, we can benefit from the VISION Cloud management models [12], especially the resource model. The resource model provides a framework for specifying all the various features of storage systems in a uniform way. For example, the resource model specifies several technical properties of the federation participants:

- the cloud provider (e.g., Amazon) the type of store (e.g., S3 Blob Store);
- public and private cloud offerings;
- replication factor if the storage internally replicates data;
- technical properties about disk accesses (e.g., high I/O SSD storage);
- latency of data center for locations;
- the structure of the pricing scheme for storage, usually based upon cost on per GB/month, incoming and outgoing data transfer, the number of transactions and others.

The model follows a hierarchical approach from cloud to node level that allows attributes to be defined on different levels.

4.3 Requirements Model

A requirements model is required to properly capture and structure the requirements emerging from application attributes modeling and user needs. Users should be enabled

to specify *non-technical* requirements in order to relieve them from technical aspects. Again, we rely on the VISION Cloud User Requirements and Resource Requirements Model as part of the developed management models [12]. The requirements model mainly covers the user requirements in a formalized way. Requirements are high-level criteria and mainly characterize the application data and needs for an appropriate storage service. Indeed, those requirements have to be translated to the more low-level storage characteristics given by the resource model somehow.

The following are typical criteria a user can use to specify for creating containers or objects:

- storage cost (at certain granularities);
- degree of confidentiality;
- level of reliability (possibly requesting replication);
- access speed (requiring fast disk access and low latency);

Further criteria can be found in [12]. In fact, these criteria have an indirect impact on the placement of objects and/or containers in a storage system. The requirement model allows a user to specify his demands in terms he/she is familiar with, whereupon the infrastructure is enabled to define a specific strategy regarding placement of data to satisfy these demands.

4.4 Administrative Tasks to Set Up a Mapping

A user uses the requirements model to define the usage-related criteria – independently of the physical resource model of CSs. In fact, these criteria have to be somehow mapped to the physical storage properties of the resource model. This enables mapping high-level user requirements, described as metadata, to low level resource requirements and thus forms the basis to find an appropriate storage system fulfilling the specified user criteria. The important new task is to map the specified user criteria to the technical storage system parameters in such a way that a storage system fitting best to the criteria will be found. This task is done within the federation service by an appropriate mapping approach.

Similar to the general approach of VISION Cloud, an administrator of the system should perform the configuration of the system using the same REST/JSON-based approach, which is basically defined by CDMI. The already available VISION Cloud federation service provides some generic configuration REST calls that can easily be extended by a PUT operation to let an administrator submit configurations.

The basis for determining appropriate Storage Systems (CSs) for given user requirements is a list of storage systems with their properties. Hence, in a first administrative step, each storage system is described by certain categories according to the resource model: public/private, a redundancy factor, its location, access speed, latency etc.

Table 1 gives an example of such a specification in a tabular manner. Each row specifies a CS the properties of which are defined in various columns such as the type, provider, typical access speed, redundancy factor, public vs. private Cloud etc.

Similar to Fig. 4, the payload now obtains the access properties (url, port etc.) and also the properties of a participating CS, as shown in Fig. 5.

Table 1. Table RESOURCE – storage system properties [14].

Storage system	Type	Public	Provider	Redundancy factor	Access speed	Location
CS1	SSD drive	No	Samsung	1	1 ms	Local
CS2	Vision store	No	Vision	2	10 ms	Local
CS3	Blob store	Yes	Amazon	3	5 ms	Amsterdam
CS4	Table Store	Yes	Azure	3	20 ms	Dublin
CS5	Blob store	Yes	Azure	3	8 ms	Dublin

```
https://vision-tb-1.myserver.net:8080/MyTenant/vision1
{
  "storage system" : "CS1",
  "url" : "vision-tb-1.myserver.net",
  "port" : "8080",
  "type" : "SSD Drive",
  "provider" : "Samsung",
  "redundancy factor" : "1",
  "access time" : { "10", "ms"},
  "location": "local",
  "public_cloud" : "no"
}
```

Fig. 5. Extended federation payload.

4.5 User Requests

Using CDMI, a user can send a request to create a container or object with associated metadata requirements, i.e., the user criteria according to the requirements model, to the CCS. Such a request resembles Fig. 4, but now specifies the user criteria as metadata together with a percentage ranking, as Fig. 6 shows.

```
PUT vision-tb-1.cloudapp.net:8080
                /CCS/siemens/logicalContainer/newObject
{
  "confidentiality" : { "50", "%" },
  "reliability"     : { "20", "%" },
  "cost savings"    : { "30", "%" }
}
```

Fig. 6. PUT request for storing an object with user criteria.

In this example, a user specifies a weight of 50% for the dimension "Confidentiality" and weights of 30% for the dimension "Reliability" and 20% for the dimension "Cost Savings".

The CCS contacts the federation service to ask for the relevant federation information. The federation returns the referring storage location(s) accordingly.

4.6 Technical Details

In knowledge-based systems, the problem of finding a recommendation is usually formulated as a pair (R, E). Here, R corresponds to the set of user requirements (according to the requirements model) while E is the set of elements that form the knowledge base. That is, the features of the resource model, i.e., the entries in Table 1, are the elements E.

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

$$\forall e_i \in S : e_i \in \sigma_R(E)$$

$\sigma_R(E)$ should here denote a selection restricting the elements in E to those that satisfy R. For instance, suppose the user requirements are defined by the concrete set:

$$R = \{r_1 : \text{access speed} \leq 15, r_2 : \text{public} = \text{yes}\}$$

Obviously CS5 is the only storage system that satisfies such a requirement R.

Table 2. Mapping table MAP [14].

	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>=30 ms			8	
access speed<30 ms			6	
access speed<20 ms			4	
access speed<10 ms			1	
…				

The missing step is the procedure for recommending appropriate storage systems. In order to recommend appropriate systems for the user storage request, a Multi-Attribute Utility Theory (MAUT) is used. In general, a MAUT schema allows

assessing and ranking a user's resource requirements according to the dimensions of a particular interest. The dimensions of interest are the user requirements such as reliability or cost savings. To set up a MAUT schema, the individual property values of the storage systems are related to the dimensions of interest with a certain ranking ranging from 1 to 10; a value of 10 means highly relevant. Consequently, each entry of the MAUT schema contains a value that defines the relevance for the related dimension by associating a certain weight. The larger the value is, the more the property contributes to the dimension of interest.

Table 2 presents a sample MAUT schema. Each row represents an individual property value, i.e., each facet of the user requirements model like public is split into several rows according to the number of values. In order not to flood the table, comparisons like '`redundancy factor>2`' are useful, especially if a finer classification is not reasonable.

The content of Table 2 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 quite low weight of 1 because a public Cloud infrastructure cannot be considered confidential;
- `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;
- `reliability` with a weight of 2;
- `cost savings` with a weight of 2
- etc.

Analogously, the `redundancy factor`, more precisely the corresponding values or ranges 1, 2 and >2, has an impact on the reliability (increasing with the factor), to the cost savings (decreasingly) etc., but none on confidentiality (empty cells).

4.7 Determining Recommendations

Based upon this data, the recommendation takes place for a given user request like the one in Fig. 6. Indeed, a request U formulates a user's request including a ranking. Finding adequate storage systems implies that these requirements have to be satisfied.

To formalize the approach, we introduce the following notions:

- `WEIGHT(U,d)` corresponds to a user request and should specify the percentage for a request U for each dimension d of interest in a request. For example:
  `WEIGHT(<request of Fig. 6>,"confidentiality") = 50`
- `RESOURCE:` $StorageSystems \times Properties \rightarrow Values$ is function reflecting the contents of Table 1. As an example, `RESOURCE("CS1", "public")` calculates to the value `"no"`.
- `MAP:` $Properties \times Values \times$ `Dimensions` $\rightarrow$ `Int`

represents the MAUT mapping table. For example, `MAP("public","yes","Reliability")` refers to the first line in Table 2 and computes to the value 5.

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

$$\texttt{Relevance}(\texttt{U},\texttt{CS}_\texttt{i}) = \sum_{\texttt{d}\in\texttt{Dimensions}} \Big(\sum_{\texttt{p}\in\texttt{Properties}} \texttt{MAP}(\texttt{p},\texttt{RESOURCE}(\texttt{CS}_\texttt{i},\texttt{p}),\texttt{d}) \times \texttt{WEIGHT}(\texttt{U},\texttt{d})\Big)$$

Using this formula, for each CSi a value for each dimension is calculated. Table 3 illustrates the calculation for our example. The line for CS1 should be understood as follows: `Confidentiality` is only determined by '`public=yes`' and '`public=no`'. That is CS1 and CS2 obtain a value of 9 because of '`public=no`', while the others have a value of 1. `Reliability` is affected by '`public=`', '`redundancy factor=`', and '`provider=`'. Since CS1 has the properties '`public=no`', '`redundancy factor=1`', and '`provider=Samsung`', the value computes to 2 + 1 + 5 according to Table 2.

Table 3. Sample calculation [14].

Storage system	Confidentiality [50%]	Reliability [20%]	Cost savings [30%]	Overall result
CS1	9	2 + 1 + 5	2 + 9 + 3 + 1	10.6
CS2	9	2 + 3 + 8	2 + 5 + 5 + 4	11.9
CS3	1	5 + 9 + 7	8 + 1 + 5 + 1	9.2
CS4	1	5 + 9 + 7	8 + 1 + 4 + 6	10.4
CS5	1	8 + 1 + 1	8 + 1 + 4 + 1	8.9

The last column in Table 3 takes the values and weights them with the relevance percentage for each domain of interest. The storage CSi with the highest overall value is suited best to satisfy the demands. That is, CS2 having a value of 11.9 is best suited for the given set of requirements while CS5 with a value of 8.9 is the worst choice.

4.8 Properties of the Approach

In spite of the examples being kept intentionally simple, the potential for users is high. For example, the price schemes of Cloud providers are quite complex, taking into account various factors such as type of Virtual Machine, in/outgoing data transfer, storage capacity (provisioned or used), and transaction rates. In fact, these all are rather technical properties of a (Cloud) storage system (in the sense of a resource model, cf. Sect. 4.2), which can be rated and mapped to the "Cost Savings" user criterion according to what aspect dominates the costs. Due to our knowledge, complex price schemes cause problems for users to estimate the overall costs.

In general, satisfying a set of requirements might lead to a conflict solving all the requirements. For example, a request

$$\mathtt{R} = \{\mathtt{r}_1 : \mathtt{public} = \mathtt{no}, \mathtt{r}_2 : \mathtt{redundancy\,level} >= 2, \\ \mathtt{r}_3 : \mathtt{access\,speed} \leq 10\}$$

does not possess a solution for the data in Table 1, i.e., $\sigma_{\mathtt{R}}(\mathtt{E}) = \emptyset$, or in other words, R raises a conflict. A relaxation $\mathrm{R}_{\mathrm{relax}}$ of R, e.g., abstaining from `r1`, could lead to a solution. Indeed, such a relaxation should be minimal in the sense that so subset $\mathrm{R}'_{\mathrm{relax}} \subseteq \mathrm{R}_{\mathrm{relax}}$ exists with $\sigma_{\mathrm{R}'\mathtt{Relax}}(\mathtt{E}) \neq \emptyset$. There is a typical problem of finding those relaxations in an efficient manner.

The advantage of our approach lies in the fact that no conflicts can occur compared to [29]. Indeed, conflicts might occur at the lower, physical level of resources, as described above. However, users are not directly formulation requirements at that lower level. Instead, more abstract requests are formulated according to the requirements model. Even if a request would equally rate all requirements like confidentiality, reliability, and cost savings with 33.3% or any other percentage, no conflict could arise: There will always be a compromise found. Using percentages, the user is able to control some preferences. Hence, no conflict detection is required as in other recommendation systems.

As a disadvantage, the user is unable to declare a physical property such as `private=yes` as an absolute must.

4.9 Monitoring and Recommendation System

There are two challenges related to the multi-criteria approach:

1. The approach relies on the correctness of the storage properties as being fed into the system. However, the specified values might be wrong or might change over time. As an example, popularity of videos might change over time leading to different optimal storage characteristics.
2. The user specifies his demands on storage, but he might have a different perception about his usage scenarios.

To tackle both points, we have set up an advanced monitoring and recommendation system. The task of the monitoring system is to collect all metrics that can be used to optimize the placement of data, both on the application as well as on the hardware level.

Interesting metrics include:

- number of accesses to an object;
- access speed;
- geographic location of requests;
- downtime (recognized by failures returned by a storage system);
- costs (either calculated online according to price schemes or using the Cloud provider's bill).

4.9.1 Monitoring System Architecture

The metrics described above should be collected from the different sources that can be physically distributed and propagated to a higher level where decision making components can process them. In order to do so effectively, it is best if information is aggregated as much as possible on the lowest level. Therefore the developed monitoring system follows a distributed model. Similar to the CSS, a monitoring instance runs on every node of the VISION Cloud. The instance is comprised of:

- collectors, for collecting hardware as well as application level metrics (such as access requests to the CCS component),
- a rule engine that is used to evaluate every event received and to trigger its preprocessing or immediate dispatching to consumers,
- an aggregator that can aggregate and process events over specific time-windows, and
- a dispatcher that is responsible for communicating relevant events to consumers using a high performance messaging library (ZeroMQ [30]).

The rules format is consisted of three sections:

- the aggregation period
- a "when" section specifying basic Boolean operation on the fields of the events, and
- a "then" section specifying the action to be taken in the form of a function.

An example of a rule that sums all the bytes read from objects belonging to a specific tenant over a 10 min span can thus be:

```
agg_period=10
when
  (e.op = 'read') AND (e.tenant = 'sie')
then
  SUM e.bytes
```

By aggregating this information on the lowest possible level we avoid unnecessary network communication at the expense of processing power. The aggregated information from multiple nodes can in turn be collected on a higher level where complex events can be produced and consumed by the recommendation system.

When an external system is federated with a VISION Cloud system, a dedicated component uses the APIs exposed by the federated system to collect relevant information and propagate this to the local representative of the monitoring system. Information is thus collected in an efficient manner from all the nodes of the system and then stored in a distributed database. This information can be used in different ways.

4.9.2 Usage of the Monitoring and Recommendation System

Firstly, collecting this information and analyzing it enables the system to adjust the storage properties to reality. Such an approach relieves the user from explicitly giving feedback. Due to our experience, a user often feels unable (and/or annoyed) to judge the quality of a service, i.e., how good his storage request is satisfied. Hence, an automatic approach seems to be more appropriate.

Secondly, the user requests are also checked by the monitoring system and recommendations can be made. For example, a user request might have given for a specific container or dedicated objects fast access a higher priority than costs for the high quality storage. However, the monitoring component might detect that these objects are accessed only sporadically. Obviously, the user had a wrong understanding of his accesses at the time s/he specified the request properties. The recommendation system is able to help in such a situation by notifying "your original storage request asked for fast access for the following objects, however, you are accessing those objects only once a week. We recommend moving the objects to a cheaper location by changing your storage request properties to the following: …" even if cost savings did not have a high priority in the original storage request. Similarly, assume cost savings are relevant for the user. The recommendation system is able to detect cheaper storage solutions by monitoring accesses and verifying the other specified storage parameters.

Although the idea is quite straightforward, we detected some challenges during the implementation. At first, any implicit modification of storage parameters made by the monitoring component – requires a re-location, not only of the directly affected users' objects but also all the other objects and containers. Indeed, the location of other objects also relies on valid storage parameters. Such a re-location is required for manually adjusted (by the administrator) because of known changes as well. Also, adding a new storage system node to the federation might have a similar impact on the distribution of objects to storage systems. Thus, such changes might result in a chain of transfer operations that would require prediction of the resulting load and effort put onto the federation. There are some further negative implications. Any movement of data might cause additional cost depending on the cloud storage solution used and presumably affect the availability of data during the relocation process. In particular, public Clouds charge for in- and outgoing data transfer, which could sum up to enormous cost in case of larger data volumes. Consequently, the cost saving criterion has to be re-considered in more detail in order to reveal the transfer costs. The same holds for user recommendations if cheaper storage locations are suggested.

4.10 Automatic Adjustment of Mappings

The presented approach not only recommends a storage system for particular requirements, but also immediately uses the best system – as decided by the recommendation component – for storing containers and objects. This works fine if the mapping table (cf. Table 2) is correct.

We are currently considering an alternative to let users affect the taken decision for a storage system. Consequently, the recommendation system only *suggests* a storage system, which can afterwards be *overridden* by a user by choosing another system while still keeping the original requirements. Since the user has obviously found a more suitable storage system for his/her needs, the mapping table does not seem to be optimal.

To improve the situation, we pursue the goal is to automatically adapt the mapping table in such a way that the selection preferred by the user would have been obtained. To this end, we applied a heuristic in a first version that is currently under evaluation and will be improved in the future.

Let us continue the example in Table 3 and let us assume that the user prefers CS1 instead of CS2 as suggested by the recommendation component. In a first step, the heuristics computes the difference of results

$$\texttt{Delta(CS2,CS1)} = \texttt{Result(CS2)} - \texttt{Result(CS1)} = 1.3$$

between both storage systems. If the difference is larger than a configurable threshold, an adjustment does not seem to be reasonable and will thus be renounced.

Otherwise, a second step finds the differences of the storage systems in the REOSURCE table (Table 1). In case of CS1 and CS2, both systems possess the same properties `Public=no` and `Location=local`. It is not reasonable to change these two so that only the remaining properties are potential candidates for adjustments. The number of candidates is thus limited.

In principle, we have to find new values for those remaining candidates in the mapping table such that $\texttt{Result}_{\texttt{new}}(\texttt{CS2}) - \texttt{Result}_{\texttt{new}}(\texttt{CS1}) < 0$, but as small as possible. Thereby, $\texttt{Result}_{\texttt{new}}$ should refer to the new calculation of the result according to the adjusted values. This is a solvable problem.

In our example, we obtain the following candidates for adjustments as highlighted in bold type in Table 4.

Table 4. Candidates (Color table online)

Storage System	Confidentiality [50%]	Reliability [20%]	Cost Savings [30%]	Overall Result
CS1	9	2 + **1** + **5**	2 + **9** + **3** + **1**	10.6
CS2	9	2 + **3** + **8**	2 + **5** + **5** + **4**	11.9

The bold numbers in Reliability refer to redundancy factor and provider, while the bold numbers in Cost Savings are calculated based upon redundancy factor, provider, and access speed. Concerning Reliability, any increment by 1 in CS1 (or decrement by 1 in CS2, respectively) reduces the difference `DELTA` by 0.2, while any increment by 1 within Cost Savings reduces the difference by 0.3. Obviously, we have the following options to bridge the gap of `DELTA = 1.3`:

(a) Reliability + 7 => 1.4
(b) Reliability + 6, Cost Savings + 1 => 1.5
(c) Reliability + 4, Cost Savings + 2 => 1.4
(d) Reliability + 3, Cost Savings + 3 => 1.5
(e) Reliability + 1, Cost Savings + 4 => 1.2
(f) Cost Savings + 5 => 1.5

Please note that Reliability + 5, Cost Savings + 2 is not an option as c) requires lower values, and similar for Reliability + 2, Cost Savings + 4. Within each group, there are again several options to realize the +x. For instance, (Reliability + 1) as part of (e) can be achieved for CS1either by

- increasing the value for "redundancy factor=1" by 1 or
- increasing the value for "provider=Samsung" by 1

in order to provide a better ranking for CS1, or the other way round by decreasing the values analogously for redundancy factor=2 and provider=Vision for CS2. Similarly, combinations for cost savings can be derived.

In a final validation step, the combinations are used to calculate for the most recent recommendations in order to check what combination of new values would have falsified previous recommendations done in the past. Only if none of the recent recommendations suggests a different storage solution, the adjustment will become effective.

This approach seems to be promising despite being simple to realize in an efficient manner.

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 [24] defined 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". They described 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).

The MetaStorage system [3] 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.

Based upon Google App Engine, Bunch et al. [6] 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. [1]. 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. [4] 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 was also one of storage layer options that VISION supports.

There is a lot of ongoing work in the area of multicloud APIs and libraries. Their goal is also to enable a unified access to multiple different cloud storage systems. Among them, Apache Libcloud [2], Smestorage [25] and Deltacloud [9] 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 referring to the metadata. 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. [5] 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, this does not represent a storage system in the sense of database federations. As CDNs in general, Meta CDN mostly focuses on read performance and neglects write performance and further criteria as well. 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 [19] 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 gathers and encrypts 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 multilevel approach, Nasuni is a migration approach that eventually moved data to the public cloud. Another example is Nimbula [20], 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 [8] 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 [21]. Nirvanix offers a private cloud on premises to their customers, and enables data transfer to the public Nirvanix Cloud Storage Network. However, 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.

6 Conclusions

Nowadays, several storage solutions are available, no matter whether traditional relational databases, deployed on premises or in a Cloud, other Cloud offerings like blob or table stores, or more recent technologies such as NoSQL databases [22]. Obviously, each technology and even more each individual product has virtues of its own, let it be specific functionality like advanced redundancy concepts or other aspects like costs or performance.

This paper tries to tackle such heterogeneity by a multi-criteria federation approach. The approach allows using several storage technologies with different properties for different purposes in parallel, particularly finding an appropriate storage solution according to individual requirements. A federation concept is used to offer a uniform access interface to several heterogeneous systems, thereby leaving the participating systems widely autonomous [24].

The presented approach is based upon the VISION Cloud software stack [27]. The VISISON Cloud project focused on efficiently storing large object, which are available in many domains: analysis images in the medical domain or Virtual Machine images in the Cloud computing area. VISION Cloud has been chosen as a basis because it offers an interesting and appealing metadata concept as an integral part for handling storage entities: Metadata can be attached to containers (including large objects and other containers) and objects to enable an efficient retrieval in first place. However, the metadata concept is also useful for controlling data placement. The key idea is to attach storage requirements as metadata as well to containers and objects that an appropriate storage solution should satisfy. The storage requirements are specified at a higher semantic level instead of more technical physical properties.

The overall approach has been described in detail focusing on what VISION Cloud concept can be reused and what additional component is required. In general, VISION Cloud and its storage system architecture are well-suited to support a multi-criteria approach. The major component of mapping user requirements to more physical storage parameters can easily be integrated into the VISION Cloud architecture.

Our future work will focus on evaluating the effectiveness of the multi-criteria approach and its included monitoring and recommendation system. In particular, we want to investigate in more detail how effective users can be shielded from complex price schemes of public Cloud providers, i.e., whether reasonable mappings from price aspects to a cost user criterion can be found. Moreover, 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 partially received funding from the European Union's Seventh Framework Programme (FP7 2007-2013) Project VISION Cloud under grant agreement number 217019 and is accordingly based on its deliverables and research publications as citations have pointed out.

References

1. Abu-Libdeh, H., Princehouse, L., Weatherspoon, H.: RACS: a case for cloud storage diversity. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 229–240. ACM, New York (2010)
2. Apache Libcloud: a unified interface to the cloud. http://libcloud.apache.org/. Accessed 15 Nov 2017
3. Bermbach, D., Klems, M., Tai, S., Menzel, M.: Metastorage: a federated cloud storage system to manage consistency-latency tradeoffs. In: Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011, pp. 452–459. IEEE Computer Society, Washington, DC (2011)
4. Brantner, M., Florescu, D., Graf, D., Kossmann, D., Kraska, T.: Building a database on S3. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, p. 251 (2008)
5. Broberg, J., Buyya, R., Tari, Z.: Creating a 'Cloud Storage' mashup for high performance, low cost content delivery. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 178–183. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01247-1_17
6. Bunch, C., et al.: An evaluation of distributed datastores using the appscale cloud platform. In: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010, pp. 305–312. IEEE Computer Society, Washington, DC (2010)
7. CDMI: Cloud data management interface version 1.1.1. https://www.snia.org/sites/default/files/CDMI_Spec_v1.1.1.pdf. Accessed 15 Nov 2017
8. CloudSwitch Homepage. http://www.cloudswitch.com. Accessed 15 Nov 2017
9. Deltacloud Homepage. http://deltacloud.apache.org/. Accessed 15 Nov 2017
10. Fielding, R., Taylor, R.: Principled design of the modern web architecture. ACM Trans. Internet Technol. **2**(2), 115–150 (2002)
11. Fox, A., et al.: Above the clouds: a Berkeley view of cloud computing. Report UCB/EECS, 28, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley (2009)
12. Gogouvitis, S.V., Katsaros, G., Kyriazis, D., Voulodimos, A., Talyansky, R., Varvarigou, T.: Retrieving, storing, correlating and distributing information for cloud management. In: Vanmechelen, K., Altmann, J., Rana, O.F. (eds.) GECON 2012. LNCS, vol. 7714, pp. 114–124. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35194-5_9
13. Hohenstein, U., Jaeger, M., Dippl, S., Bahar, E., Vernik, G., Kolodner, E.: An approach for hybrid clouds using vision cloud federation. In: 5th International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing), Venice, pp. 100–107 (2014)
14. Hohenstein, U., Jaeger, M., Gogouvitis, S.: A multi-criteria approach for large-object cloud storage. In: 6th International Conference on Data Science, Technology and Applications (DATA 2017), Madrid, Spain, pp. 75–86 (2017)
15. Jaeger, M.C., Messina, A., Lorenz, M., Gogouvitis, S.V., Kyriazis, D., Kolodner, E.K., Suk, X., Bahar, E.: Cloud-based content centric storage for large systems. In: Proceedings of Federated Conference on Computer Science and Information Systems - FedCSIS 2012, Wroclaw, Poland, 9–12 September 2012, pp. 987–994 (2012)
16. Kolodner, E., et al.: A cloud environment for data intensive storage services. In: CloudCom, pp. 357–366 (2011)
17. Kolodner, E., et al.: Data intensive storage services on clouds: limitations, challenges and enablers. In: Petcu, D., Vazquez-Poletti, J. (eds.) European Research Activities in Cloud Computing, pp. 68–96. Cambridge Scholars Publishing, New Castle upon Tyne (2011)

18. Mell, P., Grance, T.: The NIST definition of cloud computing. NIST special publication 800-145, Sept 2011
19. Nasuni Homepage. http://www.nasuni.com/. Accessed 15 Nov 2017
20. Nimbula Homepage. http://en.wikipedia.org/wiki/Nimbula. Accessed 15 Nov 2017
21. Nirvanix Homepage. http://www.nirvanix.com/products-services/cloudcomplete-hybrid-cloud-storage/index.aspx. Accessed 15 Nov 2017
22. NoSQL databases. http://nosql-database.org. Accessed 15 Nov 2017
23. Sandalage, P., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, Upper Saddle River (2013)
24. Sheth, A., Larson, J.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. **22**(3), 183–236 (1990)
25. SmeStorage Homepage. https://code.google.com/p/smestorage. Accessed 15 Nov 2017
26. Vernik, G., et al.: Data on-boarding in federated storage clouds. In: Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD 2013, pp. 244–251. IEEE Computer Society, Washington, DC (2013)
27. VISION-Cloud project consortium: high level architectural specification release 1.0, vision cloud project deliverable D10.2, June 2011. http://www.visioncloud.com. Accessed 15 May 2017
28. VISION-Cloud project consortium: data access layer: design and open specification release 2.0, deliverable D30.3b, Sept 2012. http://www.visioncloud.com/. Accessed 15 May 2017
29. Junker, U.: QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 2004), pp. 167–172 (2004)
30. ZeroMQ. http://zeromq.org/. Accessed 15 Nov 2017