Octopus
A Redundant Array of Independent Services (RAIS)

Christian Baun[1], Marcel Kunze[2] and Denis Schwab[3] and Tobias Kurze[2]
[1]*SAP AG, Walldorf, Germany*
[2]*Steinbuch Centre for Computing, Karlsruher Institut für Technologie, Eggenstein-Leopoldshafen, Germany*
[3]*Hochschule Karlsruhe, Fakultät für Informatik, Karlsruhe, Germany*
christian.baun@sap.de, marcel.kunze@kit.edu, schwab.denis@gmail.com, tobias.kurze@kit.edu

Keywords: Cloud Computing:Cloud Federation:Storage Services:Availability.

Abstract: Cloud storage services such as the Amazon Simple Storage Service (S3) are widely accepted and are reaching an ever-expanding range of customers. Especially services providing S3-compatible interfaces enjoy great popularity due to S3's simplistic, yet powerful approach to store and retrieve data via web protocols. While cloud storage services present a convenient tool, they also might turn into a risk for your data. Apart from planned service outages which may or may not be covered by Service Level Agreements (SLA), there is no guarantee that a service provider might go out of business. One might also imagine that data could be destroyed, lost or altered due to unplanned outages or physical disaster. One possibility to improve availability and also data robustness is to consume services of more than one cloud storage provider simultaneously and to establish a federated, redundant cloud storage system. Octopus cloud storage implements such a federated system realizing the concept of a Redundant Array of Independent Services (RAIS).

1 INTRODUCTION

Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) are the most popular cloud service delivery models. Since Amazon launched its Simple Storage Service (S3) to handle web objects in 2006, an ecosystem of compatible tools and implementations came into existence. While the functionality of object-based storage services usually is sufficient for many use-cases, the accompanying dependency to a provider may present a problem. If a provider goes out of business, nor the service and neither the stored data would be accessible any more. A federated cloud service setup, based on the simultaneous use of multiple providers may be more resilient. In addition, a high availability setup may be realized comprising multiple, independent services.

This paper highlights the concept of a Redundant Array of Independent Service (RAIS) and uses federated object-based storage services to implement a prototype called Octopus.

In section 2 we provide some background information on object based cloud storage services and their issues regarding availability. We also give a short introduction concerning the concept of cloud federation and how our redundant storage service conceptually fits into the picture. Section 3 outlines the concept of storage federation and the redundant storage service. Section 4 explains the service architecture, and describes some challenges and issues we faced and their corresponding solution. Finally, in section 6 we conclude our work be giving an overview of potentially useful extensions, as well as of persisting challenges we did not tackle yet.

2 BACKGROUND

2.1 Object-based Storage Services

The Amazon Web Services (AWS)[1] are a popular collection of public cloud service offerings. Due to their widespread use, the AWS API may be considered as a de-facto standard for cloud services. Amongst others, the AWS include S3, a web object data storage for applications. The AWS comprise a broad range of additional cloud computing web services like Elastic Block Store (EBS), Relational Database Service (RDS), SimpleDB, and DynamoDB.

[1] http://aws.amazon.com

Table 1: Interfaces of S3 compatible Cloud storage services

	REST	SOAP	BitTorrent
Amazon S3	X	X	X
Google CS	X	—	—
Host Europe CS	X	—	—
Eucalyptus Walrus	X	X	—
OpenStack Swift	X	—	—
Nimbus Cumulus	X	—	—

S3 stores data in form of web objects and each object is assigned to a bucket. Objects are identified by unique keys and are addressable using HTTP URLs. S3 implements a flat name space and has therefore no support for folder hierarchies. Buckets and objects may be created, listed and retrieved using web service interfaces either via REST or SOAP. Objects may also be retrieved using HTTP GET and a BitTorrent protocol interface. Users may check and alter the access rights – the Access Control List – of each S3 bucket and object to authorize access to the data.

2.1.1 S3 Ecosystem

An advantage of S3 is the growing ecosystem of compatible programming libraries, management tools and services. With Walrus from Eucalyptus[2] (Nurmi et al., 2008a) (Nurmi et al., 2008b) (Nurmi et al., 2009), Cumulus (Bresnahan et al., 2011) from Nimbus[3] (Keahey et al., 2009) (Marshall et al., 2010) and Swift from OpenStack[4] three open source private cloud solutions exist that implement at least a subset of the S3 API. A unique characteristic of this API in contrast to others is the existence of open source private cloud solutions and several public cloud implementations. Beside Amazon S3, Google Cloud Storage (CS)[5] and Host Europe Cloud Storage[6] provide more or less equivalent functionality and implement the S3 API as well. Table 1 gives an overview of S3 compatible cloud storage offerings.

Other object storage service offerings from, e.g., GoGrid[7] or FlexiScale[8] come with their own proprietary APIs. However, those offerings lack an ecosystem of compatible tools and open source implementations when compared to S3.

There are also efforts to establish standardized, provider-independent cloud APIs such as the Open Cloud Computing Interface (OCCI)[9] which would certainly by beneficial but currently there is only limited to no support by commercial cloud providers. As service providers usually try to seal off their offerings from competitors (Harmer et al., 2010), a multi-provider agreement for implementing an open cloud service interface is unlikely in the near future.

For the AWS API on the other hand there is a growing number of adaptor libraries available like JetS3t[10] and boto[11] as well as tools like GSutil[12], s3cmd[13] and S3Fox[14] that help working with S3.

2.1.2 Availalablity

In the following section we discuss the availability of public and private cloud storage services:

Public Cloud providers Amazon guarantees that S3 stores data redundantly at multiple facilities of a region and offers two different redundancy options. The standard S3 redundancy option provides 99.999999999% durability based on three replicas. The reduced redundancy option replicates objects only two times, resulting in an expected durability of 99.99%[15]. Switching to reduced redundancy results in cost savings of about 45%. In both cases the SLA guarantees a minimum availability of 99.99%.

For Google's Cloud Storage the Service Level Agreement (SLA)[16] guarantees an availability of 99.99% but doesn't make any statement concerning durability. If the monthly uptime percentage falls below the guaranteed value, customers of Amazon respectively Google receive service credits for future monthly bills.

Apart from guarantees stated in SLAs, there still is a risk in case of an unplanned outage.

Private Clouds Very often private cloud deployments do not guarantee SLAs the same way public cloud offerings do. If a certain level of availability is required, it has to be provided by a thoroughly developed and operated storage service. Some private cloud storage solutions help to accomplish this goal through their design. For example, the pWalrus[17]

[2] http://open.eucalyptus.com
[3] http://www.nimbusproject.org
[4] http://www.openstack.org
[5] http://developers.google.com/storage/
[6] http://www.hosteurope.de/produkte/Cloud-Storage/
[7] http://www.gogrid.com/cloud-hosting/cloud-api.php
[8] http://www.flexiscale.com/reference/api/
[9] http://occi-wg.org
[10] http://www.jets3t.org
[11] http://code.google.com/p/boto/
[12] http://code.google.com/p/gsutil/
[13] http://s3tools.org/s3cmd
[14] http://www.s3fox.net
[15] http://aws.amazon.com/s3-sla/
[16] http://developers.google.com/storage/docs/sla/
[17] http://www.pdl.cmu.edu/pWalrus/index.shtml

project tries to improve the throughput and availability of the web objects by using more than just a single instance of Walrus in a parallel way. These storage server instances use a distributed file system (e.g. PVFS, GPFS or LUSTRE) to store the data (Abe and Gibson, 2010). A disadvantage of pWalrus and similar approaches however is that the source code of Walrus and therefore the installation itself needs to be altered, which is not always possible or desirable. Another drawback is, that the simultaneous use of public and private storage services is tedious.

In order to increase availability it seems promising to design a redundant meta storage service implementing the S3 API. Such a storage solution would enable the use of various public and private storage services simultaneously without the necessity of altering these services in any way.

2.2 Cloud Federation

In (Kurze et al., 2011) we analyzed the term cloud federation and categorized federation strategies along several dimensions. Concerning IaaS, and especially storage services we identified three basic federation strategies:

- *Replication:* data items, i.e., objects in an object-based storage, are distributed and stored in multiple locations. Amongst other advantages, this usually increases availability and decreases vendor lock-in.
- *Erasure coding:* parts of data, i.e., parts of objects in an object-based storage, are split up and stored at different locations to, for example, increase overall availability or address data privacy concerns.
- *Fragmentation:* data items are stored according to their requirements. For example, it might be necessary to store sensitive, individual-related data inside a certain country or region, but then, other data might be stored elsewhere to optimize costs.

The RAIS storage described in this paper is a basic implementation of the IaaS federation concepts of *replication* and *erasure coding*. At the moment *fragmentation* is not supported as the focus is on availability rather than compliance.

3 REDUNDANT ARRAY OF INDEPENDENT SERVICES

We propose a *Redundant Array of Independent Services (RAIS)* that is based on the simultaneous usage of multiple storage services in a *RAID (Redundant Array of Independent Disks)*-like manner to increase availability and to decrease provider dependency. The storage system supports two modes of operation:

- *RAID-1-like mode:* RAID-1 provides mirroring without parity or striping. The objects are written identically to multiple storage services. At least two S3-compatible services are required to built up a RAID-1 array. The array is operational as long as at least a single service is available, though it might be in a degraded state.

  When an object is transferred to RAIS using RAID-1, the file is subsequently sequentially relayed to the registered storage services. If a single transfer to one of storage services fails, the action is rolled back, i.e., transfers to remaining storage services is cancelled and already transferred objects are deleted. This ensures the consistency of the array.
- *RAID-5-like mode:* RAID-5 provides block-level striping with distributed parity data. Therefore, objects are partitioned in equal parts, according to the number of registered storage services minus one. If it is not possible to split the data into equal partitions, zero-bits are added to the smaller part. Then, the parity information is computed for each partition and stored on the appropriate node. To ensure that data and its corresponding parity information are not stored in the same storage service, Octopus has implemented a mechanism called *rotating-parity* whose principle is illustrated in figure 1.

  To calculate the parity information in case of three storage services the original data is split into two partitions. The first partition is stored with provider A, the second partition with provider B and provider C stores the parity information, which is being calculated by applying the XOR operator to partitions one and two. In general, the parity information is calculated by applying the XOR operator to all partitions except the one that would have been stored with the corresponding storage provider. Rotating parity guarantees, that data can always be recovered - even in the case of one storage service being unavailable. If more than one storage service is unavailable though, i.e., more than one data partition is lost, the object can't be recovered any more.

  The RAID-5 operation mode not only provides better availability, but also improves data privacy, as no single provider has an entire object stored on its premise. To further improve security, the

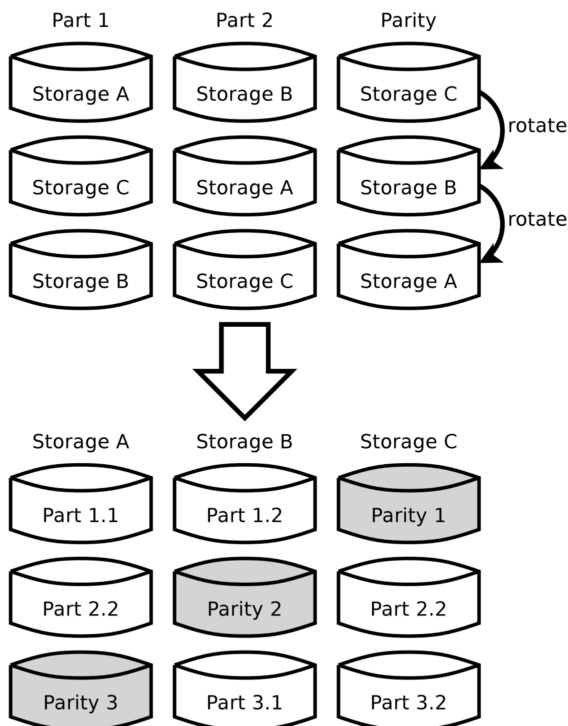

Figure 1: Uploads through Google's Blobstore

parts could also be encrypted before being uploaded. (Schwab, 2012)

To check whether objects in storage services are in sync or not, RAIS strongly relies on MD5 checksums. All S3-compatible storage services generate and store an MD5 checksum for each object. These checksums along with some other meta-data are transferred automatically whenever an object or a list of objects is requested. Figure 2 depicts the meta-data for an object with the key *"testobject.txt"*, as returned by Amazon S3, following a customer requests to list objects in a bucket. For RAIS, the important information are <Key>...</Key>, holding the display name of the object, and <ETag>...</ETag>, holding the MD5 checksum.

Whenever RAIS receives a request to list objects in a certain bucket, it checks if the data is still synchronized across the registered storage services by comparing the MD5 checksums. First, it requests the list of objects inside the user's buckets from all registered storage services. Then RAIS compares the names and checksums of the received objects inside the returned lists. If the returned lists have identical keys and checksum entries, the data is considered in sync across the registered storage services. (Baun, 2011).

```
<Key>testobject.txt</Key>
<LastModified>2012-01-29T21:18:24.000Z<
    /LastModified>
<ETag>&quot;71388
    ba9a76ddb7ecd43a14f2a9ae216&quot;</
    ETag>
<Size>2163</Size>
<Owner>
  <ID>af0af9137ff6...97272818796</ID>
  <DisplayName>username</DisplayName>
  </Owner>
<StorageClass>STANDARD</StorageClass>
```

Figure 2: Object-related meta-data as returned by Amazon S3

4 Platform & Architecture

The RAIS concept has been implemented using the Google App Engine (GAE) in a project called Octopus. GAE is a platform as a service (PaaS) offered by Google that provides a stable and highly available runtime environment at a fair price or - depending on the usage - even for free. It comes with APIs to manage user authentication, mail delivery and reception, manipulate images and many more. Another advantage of GAE is its inherent scalability. There are compatible and free private cloud alternatives to GAE, namely AppScale[18] (Chohan et al., 2009)(Bunch et al., 2010) and typhoonAE[19].

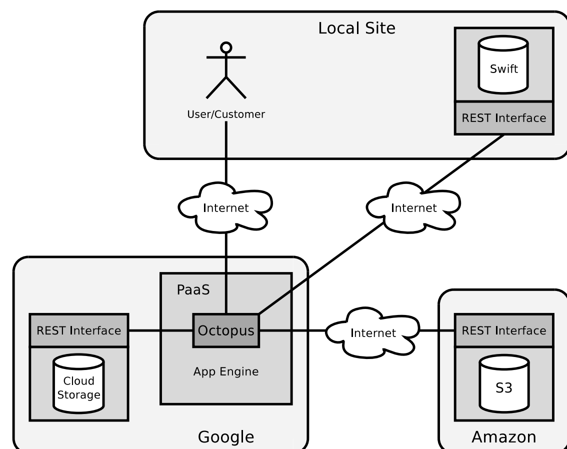

Figure 3: Octopus running inside the public cloud PaaS

Figure 3 depicts how Octopus may be run in a public cloud. In contrast, figure 4 shows the situation when operating Octopus in a private cloud environment based on AppScale respectively typhoonAE.

Octopus has been implemented in Python using

[18] http://appscale.cs.ucsb.edu
[19] http://code.google.com/p/typhoonae/

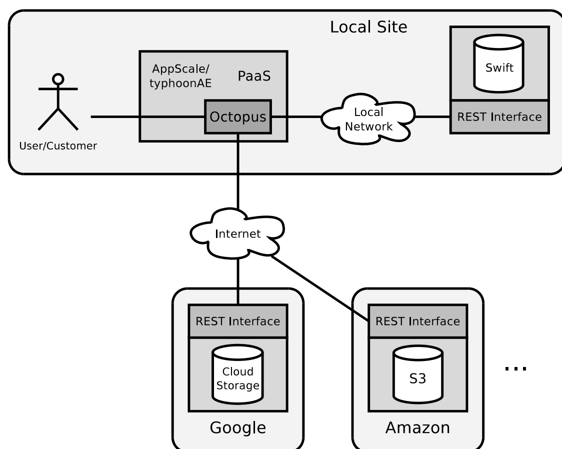

Figure 4: Octopus running inside a private cloud PaaS

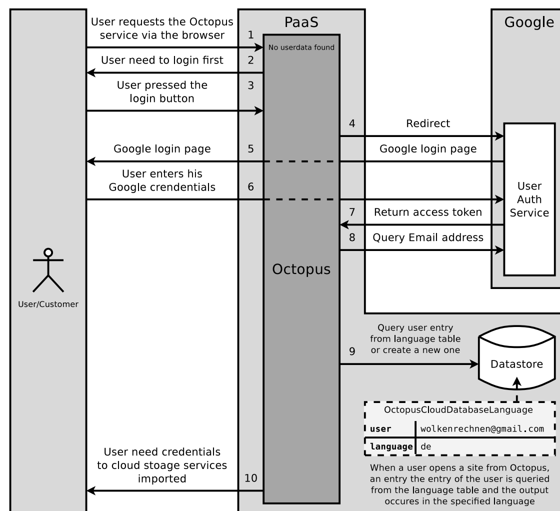

Figure 5: Concept of Octopus' user management and authentication

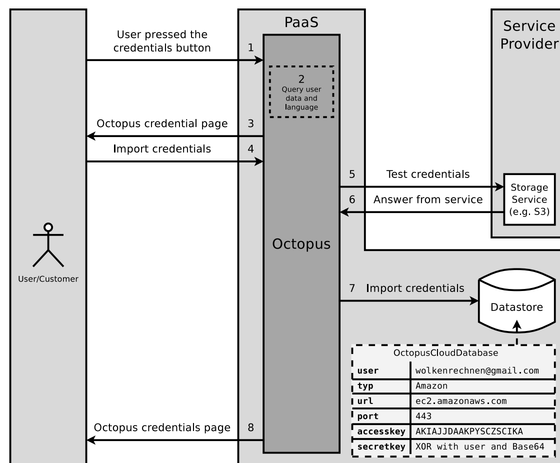

Figure 6: Concept of Octopus' managing the users credentials to storage services

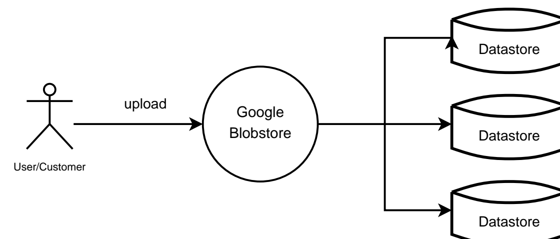

Figure 7: Uploads through Google's Blobstore

the web framework Django[20]. In order to communicate with storage services compliant to S3, Octopus uses boto, an open source python interface to AWS.

The App Engine provides several helpful APIs e.g. to use the Google authentication mechanism and storage services of the Google infrastructure. The user management and authentication of Octopus is based on Google user accounts (see figure 5). The Datastore (a BigTable service with a SQL-like query language) is used to store the customers imported credentials for S3-compatible storage services (see Figure 6). The corresponding APIs and services of the App Engine are supported by AppScale and typhoonAE too. If Octopus is running on a private AppScale or typhoonAE platform service the user authentication is implemented with AppScale respectively typhoonAE so that there is no need to use any Google user accounts.

When multiple storage services are used to store the same file or parts of a file redundantly, it is obviously necessary to transfer the file respectively the parts multiple times as well. In a first implementation of Octopus, files were uploaded directly from the clients, i.e., from the clients browsers to each associated storage service. This was an easy but very inefficient and time consuming solution.

In order to improve Octopus an object to be stored in multiple storage services is uploaded only once and cached in the Blobstore before being uploaded to the storage services. Though this is a much better approach from a customer's perspective, it also is technically more complex and has to respect restrictions of Google's Blobstore, such as the maximum file size, for example. Figure 7 illustrates how Octopus handles uploads.

[20] http://www.djangoproject.com

4.1 Interaction with Storage Services

Octopus has been designed to work in a multi-user environment. To support multiple tenants simultaneously, it prepares a bucket for each user upon his or

hers registration for a certain storage service. Therefore, the following scheme for bucket names has been deployed: `octopus-storage-<username>`. These buckets, created for every registered storage service, serve as root storage locations for Octopus. In order to register a storage service a user has to provide the corresponding credentials using the Octopus web interface.

Once a storage service is registered, Octopus interacts with its S3-compatible REST API. We are not using SOAP interfaces or BitTorrent due to limited support by some storage providers. Table 1 provides an overview of interfaces implemented by different cloud storage services.

REST is an architectural-style that relies on HTTP methods like `GET` or `PUT`. S3-compatible services use `GET` to receive the list of buckets that are assigned to an user account, or a list of objects inside a bucket or an object itself. Buckets and objects are created with `PUT` and `DELETE` is used to erase buckets and objects. `POST` can be used to upload objects and `HEAD` is used to retrieve meta-data from an account, bucket or object. Uploading files into S3-compatible services is done via HTML forms and `POST` directly from the customers client, i.e., browser. Table 2 gives an overview of methods used to interact with S3-compatible storage services.

Table 2: Description of the HTTP methods with request-URIs that are used to interact with storage services

	Account-related Operations	
GET	/	List buckets
HEAD	/	Retrieve metadata
	Bucket-related Operations	
GET	/bucket	List objects
PUT	/bucket	Create bucket
DELETE	/bucket	Delete bucket
HEAD	/bucket	Retrieve metadata
	Object-related Operations	
GET	/bucket/object	Retrieve object
PUT	/bucket/object	Upload object
DELETE	/bucket/object	Delete object
HEAD	/bucket/object	Retrieve metadata
POST	/bucket/object	Update object

The basic functionality of all S3-compatible storage services is more or less identical, but the exact behavior is often slightly different.

4.1.1 Amazon Web Services

Amazon S3 supports multi-part uploads to break larger objects into chunks of smaller size and upload a number of chunks in parallel. This feature is helpful in case of upload failures, as the upload some chunks can be restarted imposing less overhead compared to the re-upload of the entire file. So multi-part uploads could help to improve the performance of the meta-storage service but the other S3-compatible storage solutions do not support this feature and therefore it is not implemented in Octopus.

4.1.2 Cumulus and Host Europe

Cumulus and Host Europe Cloud Storage both do support uploading objects via `GET` but not via `POST` yet. Eventually future releases may have this feature to be used by Octopus. As long as this feature is missing, it is impossible to upload file objects using these services with Octopus.

4.1.3 Walrus and OpenStack

In Amazon S3, Google Cloud Storage, Cumulus and Host Europe Cloud Storage, the MD5 checksums are enclosed by double quotes (see Figure 2). In Walrus and OpenStack they are not. The reason for this difference behavior remains unclear. Octopus has been designed to handle this implementation detail and thus nonetheless works with these storage systems.

If no submit button inside a form is used to upload an object into Walrus, some bytes of garbage data is appended to the object.

An annoying issue of Walrus is that the service adheres faulty data to all objects that are transferred via `POST` when no submit element exists at the end of the HTML message. Walrus recognizes the submit element as the objects' end. If the subject element is missing, Walrus attaches all data until the end of the transfer to the object.

Another problem of Walrus is that in the version that is part of Eucalyptus 1.6, inside each bucket an object with the key `None` exists, which is incorrect. As this object cannot be erased Octopus need to ignore it.

When using Amazon S3 or Walrus, inside the HTML form and the related policy document[21], the developer has the freedom to use either die attribute `redirect` or alternatively

[21] The policy document contains i.a. the object's name, a description of the content, the bucket and the destination address to which the browser will be redirected if the upload was successful.

success_action_redirect to set the page, the browser will be redirected to, if the upload was successful. Google Cloud Storage in contrast just makes use of the attribute success_action_redirect and ignores redirect.

4.1.4 Google Cloud Storage

An issue of Google Cloud Storage is that the submit button, that is used inside a HTML form to transfer objects to the storage service must not contain the name attribute. Amazon S3 and Walrus both ignore the name attribute.

5 Related work

Various systems to avoid vendor lock-in and to improve availability have been developed. Bermbach et al. developed MetaStorage (Bermbach et al., 2011), which is a federated cloud storage system that distributes data across different cloud storage services and relies on some mechanisms from Amazon's Dynamo (DeCandia et al., 2007) for cross-provider replication to achieve higher levels of availability as well as of fault tolerance and scalability. It relies on a distributed hashtable and extends the (N,R,W)-quorum approach to include providers as a supplemental dimension. MetaStorage consists of multiple components, of these the most important ones are: storage nodes, distributors and coordinators. Storage nodes provide an abstraction from the underlying infrastructure, i.e. cloud storage services or just hard-disks. Distributors are in charge of the replication and of the retrieval of files using the underlying storage nodes. To avoid a bottlneck, multiple distributors can be used but have to be managed. Coordinators provide this management and assure that every distributor has the same configuration.

RACS (Abu-Libdeh et al., 2010) is a system that uses RAID-like techniques to distribute data across multiple cloud storage providers to, for example, reduce costs in case of a provider switch or to avoid vendor lock-in. By striping data across multiple providers, provider changes become easier; also erasure coding provides redundancy against outages. Thereby data objects are broken down into m fragments and mapped to a larger set of n fragments of the same size, in a way that the original fragments can still be obtained by certain subsets of the m-fragments. RACS is designed as a proxy between a client application and potentially different cloud providers. It provides an Amazon's S3 interface and therefore enables usage by S3-compatible software. To avoid a bottleneck, multiple RACS proxies can interact concurrently with the same storage systems. Apache ZooKeeper is used to provide distributed synchronization primitives to avoid data races.

OceanStore (Rhea et al., 2003) is another storage service that uses multiple layers of replication and distributes data over the internet to provide a persistent, und durable data store.

(Chun et al., 2006) discusses replication strategies for distributed storage systems. It focuses on the development of algorithms permitting to store data durably at low bandwidth costs. A basic implementation of the presented algorithm using distributed hash tables is presented and detailed.

HAIL (Bowers et al., 2009) is no storage system, but a system to ensure that a file, stored with storage services, is still intact and retrievable. It is a remote-file integrity check protocol that extends the principles of RAID to work in cloud settings.

(Dabek et al., 2004) provides background information about the design of an efficient distributed hash table providing high-throughput and low-latency network storage.

6 Conclusions and Future Work

We have described the concept of a Redundant Array of Independent Services (RAIS) to realize a fault-tolerant cloud storage. The application of RAIS may improves data storage availability as well as security. Octopus is a first implementation that can be described as a meta-storage service that allows to store data using storage services of different providers simultaneously.

The Octopus service is licensed as open source according to the Apache License v2.0 and the source code is available at the project site[22]. The graphical user interface currently supports the English and German language. Due to the design of the application, it is straightforward to implement additional languages.

A useful extension of Octopus would be some kind of proxy or buffer storage into which objects are uploaded by Octopus from the client side before they are distributed to the storage services. The Google Datastore is not suitable as proxy, because objects inside the Datastore can have a maximum size of 1 MB. The Blobstore is another storage service of the Google infrastructure, that can be used by applications running inside the App Engine. It allows the persistent storage of data in form of so-called BLOBs (Binary Large Objects), each with up to 2 GB. An-

[22] http://code.google.com/p/cloudoctopus/

other possible basis for a proxy is the Google Cloud Storage. Objects inside Google Cloud Storage can be accessed directly from web applications that run in the App Engine, which is a requirement for transfers via HTTP `POST` to the other storage services used. A disadvantage of the App Engine in general is that outgoing data connections are limited to 1 MB for each transfer.

As the objects are transferred directly from the customers browser to each storage service used, the amount of data that need to be transmitted increases linearly with each additional storage service. Even large objects must be transferred at least two times, if they should be stored in a redundant way. For this reason, the use of multiple storage services leads to extended transfer times.

Our next steps in the development of Octopus foresee the evaluation of further approaches to implement a proxy to upload the objects to the storage services in parallel to improve the performance.

REFERENCES

Abe, Y. and Gibson, G. (2010). pWalrus: Towards better integration of parallel file systems into cloud storage. In *Cluster10: Int. Conf. on Cluster Computing 2010*. IEEE.

Abu-Libdeh, H., Princehouse, L., and Weatherspoon, H. (2010). Racs: a case for cloud storage diversity. In *SoCC'10: 1st Symposium on Cloud computing*. ACM.

Baun, C. (2011). *Untersuchung und Entwicklung von Cloud Computing-Diensten als Grundlage zur Schaffung eines Marktplatzes*. PhD thesis, Universität Hamburg.

Bermbach, D., Klems, M., Menzel, M., and Tai, S. (2011). Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *CLOUD'11: 4th Int, Conf. on Cloud Computing*. IEEE.

Bowers, K. D., Juels, A., and Oprea, A. (2009). Hail: a high-availability and integrity layer for cloud storage. In *CSS'09: 16th Conf. on Computer and communications security*. ACM.

Bresnahan, J., Keahey, K., LaBissoniere, D., and Freeman, T. (2011). Cumulus: An open source storage cloud for science. In *ScienceCloud'11: 2nd Int. Workshop on Scientific Cloud Computing*. ACM.

Bunch, C., Chohan, N., Krintz, C., Chohan, J., Kupferman, J., Lakhina, P., Li, Y., and Nomura, Y. (2010). An evaluation of distributed datastores using the appscale cloud platform. In *Cloud10: Int. Conf. on Cloud Computing*. IEEE.

Chohan, N., Bunch, C., Pang, S., Krintz, C., Mostafa, N., Soman, S., and Wolski, R. (2009). Appscale: Scalable and open appengine application development and deployment. 1st Int. Conf. on Cloud Computing.

Chun, B.-G., Dabek, F., Haeberlen, A., Sit, E., Weatherspoon, H., Kaashoek, F., Kubiatowicz, J., and Morris, R. (2006). Efficient replica maintenance for distributed storage systems. In *NSDI'06: Symposium on Networked Systems Design and Implementation*. USENIX.

Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M. F., and Morris, R. (2004). Designing a dht for low latency and high throughput. In *NSDI'04: 1st Symposium on Networked Systems Design and Implementation*. USENIX.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. In *SOSP'07: 21st Symposium on Operating Systems Principles*, New York. ACM.

Harmer, T., Wright, P., Cunningham, C., Hawkins, J., and Perrott, R. (2010). An application-centric model for cloud management. In *SERVICES'10: 6th World Congress on Services*. IEEE.

Keahey, K., Tsugawa, M., Matsunaga, A., and Fortes, J. (2009). Sky computing. *Internet Computing, IEEE*, 13(5):43–51.

Kurze, T., Klems, M., Bermbach, D., Lenk, A., Tai, S., and Kunze, M. (2011). Cloud federation. In *CLOUD COMPUTING 2011: 2nd Int. Conf. on Cloud Computing, GRIDs, and Virtualization*.

Marshall, P., Keahey, K., and Freeman, T. (2010). Elastic Site Using Clouds to Elastically Extend Site Resources. In *CCGrid: 10th Int. Conf. on Cluster, Cloud and Grid Computing*. IEEE/ACM.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2008a). Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems. In *UCSB Computer Science Technical Report Number 2008-10*.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2008b). The eucalyptus open-source cloud-computing system. In *CCA'08: Cloud Computing and Its Applications workshop*.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. *Int. Symposium on Cluster Computing and the Grid*.

Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., and Kubiatowicz, J. (2003). Pond: The oceanstore prototype. In *FAST'03: 2nd Conf. on File and Storage Technologies*. USENIX.

Schwab, D. (2012). Implementierung eines redundanten Datenspeichers für die Hybride Cloud. *Hochschule Karlsruhe*.