Implementation of a Secure and Reliable Storage Above the Untrusted Clouds

Maxim Schnjakin, Dimitri Korsch, Martin Schoenberg, Christoph Meinel
Hasso Plattner Institute
Potsdam University, Germany
Prof.Dr-Helmertstr. 2-3, 14482 Potsdam, Germany
maxim.schnjakin, dimitri.korsch@student, martin.schoenberg@student, office-meinel@.hpi.uni-potsdam.de

***Abstract*—Cloud Computing as a service-on-demand architecture has grown in importance over the previous few years. One driving force of its growth is the ever increasing amount of data which is supposed to outpace the growth of storage capacity. This way, public cloud storage services enable organizations to manage their data with low operational expenses. However, the benefits of cloud computing come along with challenges and open issues such as security, reliability and the risk to become dependent on a provider for its service. In general, a switch of a storage provider is associated with high costs of adapting new APIs and additional charges for inbound and outbound bandwidth and requests. In this paper, we describe the design, architecture and implementation of Cloud-RAID, a system that improves availability, confidentiality and integrity of data stored in the cloud. To achieve this objective, we encrypt user's data and make use of the RAID-technology principle to manage data distribution across cloud storage providers. Our approach allows users to avoid vendor lock-in, and reduce significantly the cost of switching providers. In general, the data distribution is based on users' expectations regarding providers geographic location, quality of service, providers reputation, and budget preferences. In this paper, we also discuss the security functionality and reveal our observations on the overall performance when encrypting and encoding user's data.**

I. Introduction

Cloud Computing is a concept of utilizing computing as an on-demand service. It fosters operating and economic efficiencies and promises to cause an unanticipated change in business. Using computing resources as pay-as-you-go model enables service users to convert fixed IT cost into a variable cost based on actual consumption. Therefore, numerous authors argue for the benefits of cloud computing focusing on the economic value [10]. However, despite of the non-contentious financial advantages cloud computing raises questions about privacy, security and reliability.

Among available cloud offerings, storage services reveal an increasing level of market competition. According to iSuppli [9] global cloud storage revenue is set to rise to $5 billion in 2013, up from $1.6 billion in 2009. One reason is the ever increasing amount of data which is supposed to outpace the growth of storage capacity. Currently, it is very difficult to estimate the actual future volume of data but there are different estimates being published. According to IDC review [11], the amount of digital information created and replicated is estimated to surpass 3 zettabytes by the end of this year. This amount is supposed to more than double in the next two years. In addition, the authors estimate that today there is 9 times more information available than was available five years ago.

However, for a customer (service) to depend on solely one cloud storage provider (in the following provider) has its limitations and risks. In general, vendors do not provide far reaching security guarantees regarding the data retention. Users have to rely on effectiveness and experience of vendors in dealing with security and intrusion detection systems. For missing guarantees service users are merely advised to encrypt sensitive content before storing it on the cloud. Placement of data in the cloud removes many of direct physical controls that a data owner has over data. So there is a risk that service provider might share corporate data with a marketing company or use the data in a way the client never intended. Further, customers of a particular provider might experience vendor lock-in. In the context of cloud computing, it is a risk for a customer to become dependent on a provider for its services. Common pricing schemes foresee charging for inbound and outbound transfer and requests in addition to hosting the actual data. Changes in features or pricing scheme might motivate a switch from one storage service to another. However, because of the data inertia, customers may not be free to select the optimal vendor due to immense costs associated with a switch of one provider to another. The obvious solution is to make the switching and data placement decisions at a finer granularity then all-or-nothing. This could be achieved by replicating corporate data to multiple storage providers. Such an approach implies significant higher storage and bandwidth costs without taking into account the security concerns regarding the retention of data.

A more economical approach which is presented in this paper is to separate data into unrecognizable slices, which are distributed to providers - whereby only a subset of the nodes needs to be available in order to reconstruct the original data. This is indeed very similar to what has been done for years at the level of file systems and disks. In our work we use RAID-like (Redundant Array of Independent Disks) techniques to overcome the mentioned limitations of cloud storage in the following way:

1) **Security.** The provider might be trustworthy, but malicious insiders represent a well known security problem. This is a serious threat for critical data such as medical records, as cloud provider staff has physical access to the hosted data. We tackle the aforementioned problem by encoding and encrypting

the original data and later by distributing the fragments transparently across multiple providers. This way, none of the storage vendors is in an absolute possession of the client's data. Moreover, the usage of enhanced erasure algorithms enables us to improve the storage efficiency and thus also to reduce the total costs of the solution.

2) **Service Availability.** Management of computing resources as a service by a single company implies the risk of a single point of failure. This failure depends on many factors such as financial difficulties (bankruptcy), software or network failure, etc. However, even if the vendor runs data centers in various geographic regions using different network providers, it may have the same software infrastructure. Therefore, a failure in the software in one center will affect all the other centers, hence affecting the service availability. In July 2008, for instance, Amazon storage service S3 was down for 8 hours because of a single bit error [18]. Our solution addresses this issue by storing the data on several clouds. Whereby no single entire copy of the data resides in one location, and only a subset of providers needs to be available in order to reconstruct the data.
3) **Reliability.** Any technology can fail. According to a study conducted by Kroll Ontrack[1] 65 percent of businesses and other organizations have frequently lost data from a virtual environment. A number that is up by 140 percent from just last year. Admittedly, in the recent times, no spectacular outages were observed. Nevertheless failures do occur. We deal with the problem by using erasure algorithms to separate data into packages, thus enabling the application to retrieve data correctly even if some of the providers corrupt or lose the entrusted data.
4) **Data lock-in.** By today there are no standards for APIs for data import and export in cloud computing. This limits the portability of data and applications between providers. For the customer this means that he cannot seamlessly move the service to another provider if he becomes dissatisfied with the current provider. This could be the case if a vendor increases his fees, goes out of business, or degrades the quality of the provided services. As stated above, our solution does not depend on a single service provider. The data is balanced among several providers taking into account user expectations regarding the price and availability of the hosted content. Moreover, with erasure codes we store only a fraction of the total amount of data on each cloud provider. In this way, switching one provider for another costs merely a fraction of what it would be otherwise.

The main contributions of this paper are:

- We present an application that can be used to overcome the limitations of individual clouds by using encryption, erasure codes and by integrating various cloud storage providers;
- An set of experiments showing that given the speeds of current disks and CPUs, the libraries used (for both erasure coding and encryption) are easily fast enough to provide good performance, reliable and secure storage system;
- A performance evaluation of Cloud-RAIDs security component.

[1]http://www.krollontrack.com/resource-library/case-studies/

II. Architecture overview

The ground of our approach is to find a balance between benefiting from the cloud's nature of pay-per-use and ensuring the security of the company's data. The goal is to achieve such a balance by distributing corporate data among multiple storage providers, automizing big part of the selection process of a cloud provider, and removing the auditing and administrating responsibility from the customer's side. As mentioned above, the basic idea is not to depend on solely one storage provider but to spread the data across multiple providers using redundancy to tolerate possible failures. The approach is similar to a service-oriented version of RAID. While RAID manages sector redundancy dynamically across hard-drives, our approach manages file distribution across cloud storage providers. RAID 5, for example, stripes data across an array of disks and maintains parity data that can be used to restore the data in the event of disk failure. We carry the principle of the RAID-technology to cloud infrastructure. In order to achieve our goal we foster the usage of erasure coding technics (see III-C2). This enables us to tolerate the loss of one or more storage providers without suffering any loss of content [13].Our architecture includes the following main components:

- **User Interface Module.** The interface presents the user a cohesive view on his data and available features. Here users can manage their data and specify requirements regarding the data retention (quality of service parameters).
- **Resource Management Module.** This system component is responsible for an intelligent deployment of data based on the user's requirements.
- **Data Management Module.** This component handles data management on behalf of the resource management module.

Interested readers will find more background information in our previous work [17],[4]. The system has a number of core components that contain the logic and management layers required to encapsulate the functionality of different storage providers. The next section gives an overview on the implementation of our system on a more detailed level.

III. Design

In this section we describe how we achieve the goal of the consistent, unified view on the data management system to the end-user. The web portal is developed using Grails, JNI and C technologies, with a MySQL back-end to store user accounts, current deployments, meta data, and the capabilities and pricing of cloud storage providers. Keeping the meta data locally ensures that no individual provider will have access to stored data. In this way, only users that have authorization to access the data will be granted access to the shares of (at least) k different clouds and will be able to reconstruct the data. Further, our implementation makes use of AES for symmetric encryption, SHA-1 and MD5 for cryptographic hashes and an improved version of Jerasure library [13] for using the Cauchy-Reed-Solomon and Liberation erasure codes. Our system communicates with providers via ”storage connectors”, which are discussed further in this section.

A. *Service Interface*

The graphical user interface provides two major functionalities to an end-user: data administration and specification of requirements regarding the data storage. Interested readers are directed to our previous work [16] which gives a more detailed background on the identification of suitable cloud providers in our approach. In short, the user interface enables users to specify their requirements (regarding the placement and storage of user’s data) manually in form of options (e.g. data might be placed based on user’s price or quality of service expectations).

B. *Storage Repositories*

1) Cloud storage providers: Cloud storage providers are modeled as a storage entity that supports not more than six basic operations: $create$ a container, $write$ a data object, $read$ a data object, $list$ all data objects, $delete$ an object and $getDigest$ which returns the hash value of the specified data object. Further, the individual providers are not trusted. This means that the entrusted data can be corrupted, deleted or leaked to unauthorized parties [12]. This fault model encompasses both malicious attacks on a provider and arbitrary data corruption like the Sidekick case (section I). The protocols require $n = k + m$ storage clouds, at most m of which can be faulty. Present-day, our prototypical implementation supports the following storage repositories: Amazons S3 (in all available regions: US west and east coast, Ireland, Singapore and Tokyo), Box, Rackspace Cloud Files, Azure, Google Cloud Storage (EU and US), HP Cloud Service and Nirvanix SND. Further providers can be easily added.

2) Service repository: At the present time, the capabilities of storage providers are created semi-automatically based on an analysis of corresponding SLAs which are usually written in a plain natural language [6]. Until now the claims stated in SLAs need to be translated into WSLA statements and updated manually (interested readers will find more background information in our previous work [16]). Subsequently the formalized information is imported into a database of the system component named *service repository*. The database tracks logistical details regarding the capabilities of storage services such as their actual pricing, SLA offered, and physical locations. With this, the service repository represents a pool with available storage services.

3) Matching: The selection of storage services for the data distribution occurs based on user preferences set in the user interface. After matching user requirements and provider capabilities, we use the reputation of the providers to produce the final list of potential providers to host parts of the user’s data. A provider’s reputation holds the details of his historical performance plus his ratings in the service registries and is saved in a Reputation Object (introduced in our previous work [3], [2], [5]). By reading this object, we know a provider’s reputation concerning each performance parameter (e.g. has high response time, low price). With this information the system creates a prioritized list of repositories for each user. In general, the number of storage repositories needed to ensure data striping depends on a user’s cost expectations, availability and performance requirements. The total number of repositories is limited by the number of implemented storage connectors.

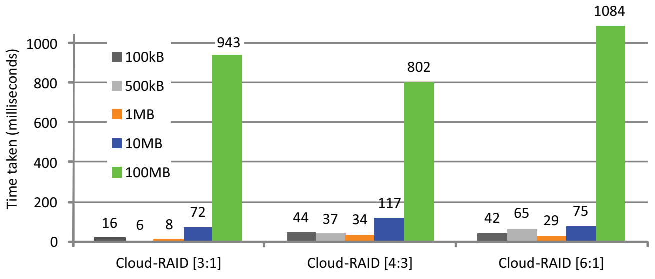


Fig. 1. The average performance of the erasure algorithm with data objects of varying sizes (100kB, 500kB, 1MB, 10MB and 100MB).

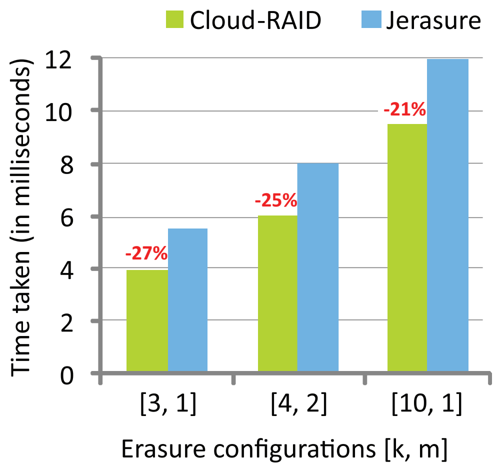
C. *Data Management*

1) Data model: In compliance with [1] we mimic the data model of Amazon’s S3 by the implementation of our encoding and distribution service. All data objects are stored in containers. A container can contain further containers. Each container represents a flat namespace containing keys associated with objects. An object can be of an arbitrary size, up to 5 gigabytes (limited by the supported file size of cloud providers). Objects must be uploaded entirely, as partial writes are not allowed as opposed to partial reads. Our system establishes a set of n repositories for each data object of the user. These represent different cloud storage repositories.

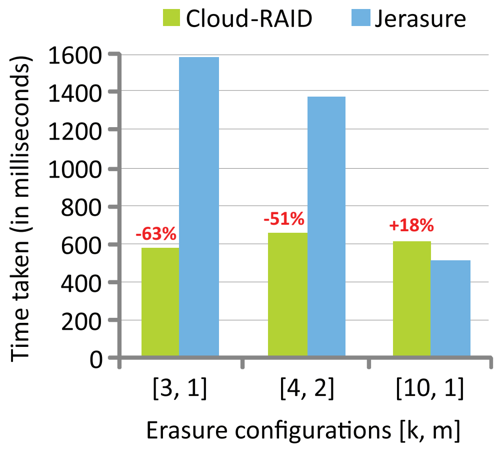
2) Encoding: Upon receiving a write request the system splits the incoming object into k data fragments of an equal size - called $chunks$. These k data packages hold the original data. In the next step the system adds m additional packages whose contents are calculated from the k chunks, whereby k and m are variable parameters [13]. This means, that the act of encoding takes the contents of k data packages and encodes them on m coding packages.

In turn, the act of decoding takes some subset of the collection of $n = k + m$ total packages and from them recalculates the original data. Any subset of k chunks is sufficient to reconstruct the original object of size s [15]. The total size of all data packets (after encoding) can be expressed with the following equation: $(\frac{s}{k}*k)+(\frac{s}{k}*m) = s+(\frac{s}{k}*m) = s*(1+\frac{m}{k})$. With this, the usage of erasure codes increases the total storage by a factor of $\frac{m}{k}$. Summarized, the overall overhead depends on the file size and the defined m and k parameters for the erasure configuration. Figure 1 visualizes the performance of our application using different erasure configurations.

In our work we make use of the Cauchy-Reed-Solomon algorithm for two reasons. First, according to [13] the algorithm has a good performance characteristics in comparison to existing codes. In their work, the authors performed a head-to-head comparison of numerous open-source implementations of



(a) Encoding of a 100kB data object



(b) Encoding of a 100MB data object

Fig. 2. Total time taken when Jerasure and Cloud-RAID libraries are used to encode data objects of varying sizes.

various coding techniques which are available to the general public. Second, the algorithm allows free selection of coding parameters k and m. Whereas other algorithms restrict the choice of parameters. Liberation Code [14] for example is a specification for storage systems with n = k + 2 nodes to tolerate the failure of any two nodes (the parameter m is fix and is equal to two). However, the functionality of the encoding component is based on the Jerasure library [13] which is an open C/C++ framework that supports erasure coding in storage applications. In our implementation we were able to improve the overall performance of the library by more than 20%. Figure 2 summarizes the results of 20 runs executed on test machine 1 (see IV-A1).

Competitive storage providers claim to have SLAs ranging from 99% to 100% uptime percentages for their services. Therefore choosing $m = 1$ to tolerate one provider outage or failure at time will be sufficient in the majority of cases. Thus, it makes sense to increase k and spread the packages across more providers to lower the overhead costs.

In the next step, the distribution service makes sure that each encoded data package is sent to a different storage repository. In general, our system follows a model of one thread per provider per data package in such a way that the encryption, decryption, and provider accesses can be executed in parallel.

However, most erasure codes have further parameters as for example w, which is word size[2]. In addition, further parameters are required for reassembling the data (original file size, hash value, coding parameters, and the erasure algorithm used). This metadata is stored in a MySQL back-end database after performing a successful write request.

3) Data Distribution: Each storage service is integrated by the system by means of a storage-service-connector (in the following service-connector). These provide an intermediate layer for the communication between the resource management service (see section III-D) and storage repositories hosted by storage vendors. This enables us to hide the complexity in dealing with proprietary APIs of each service provider. The basic connector functionality covers operations like creation, deletion or renaming of files and folders that are usually supported by every storage provider. Such a service-connector must be implemented for each storage service, as each provider offers a unique interface to its repository. As discussed earlier in this chapter all accesses to the cloud storage providers can be executed in parallel.

4) Reassembling the data: When the service receives a *read* request, the service component fetches k from n data packages (according to the list with prioritized service providers which can be different from the prioritized $write$-list, as providers differ in upload and download throughput as well as in cost structure) and reassembles the data. This is due to the fact, that in the pay-per-use cloud models it is not economical to read all data packages from all clouds. Therefore, the service is supported by a *load balancer* component, which is responsible for retrieving the data units from the most appropriate repositories. Different policies for load balancing and data retrieving are conceivable as parts of user's data are distributed between multiple providers. Interested readers will find more information about distribution strategies in our work [17].

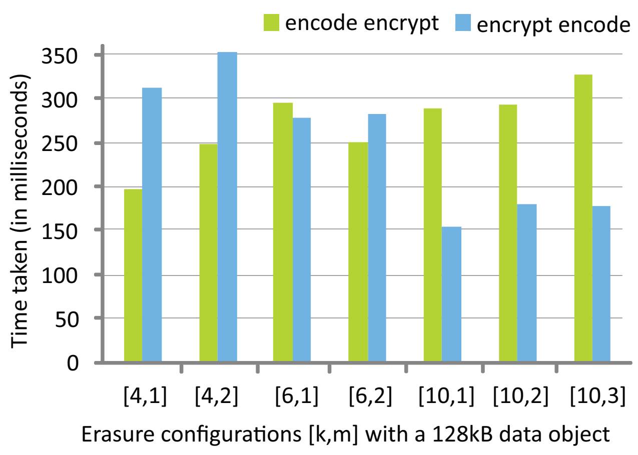
D. Resource Management Service

This component tracks each user's actual deployment and is responsible for various housekeeping tasks:

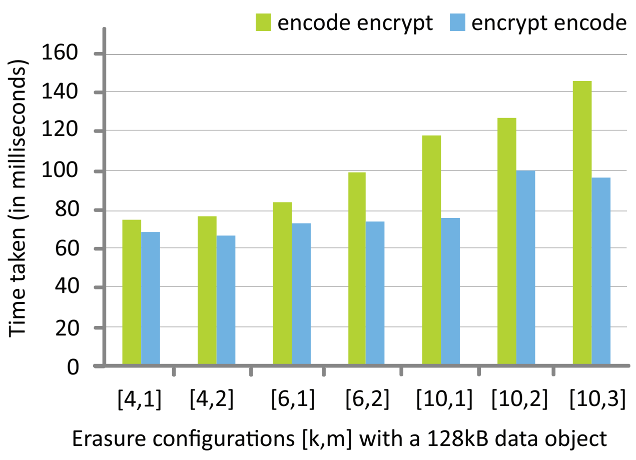
1) The service is equipped with a MySQL back-end database to store crucial information needed for deploying and reassembling of users data.
2) Further, it audits and tracks the performance of the participated providers and ensures, that all current deployments meet the corresponding requirements specified by the user.
3) The management component is also responsible for scheduling of not time-critical tasks.

Further details can be found in our previous work [17] and [5].

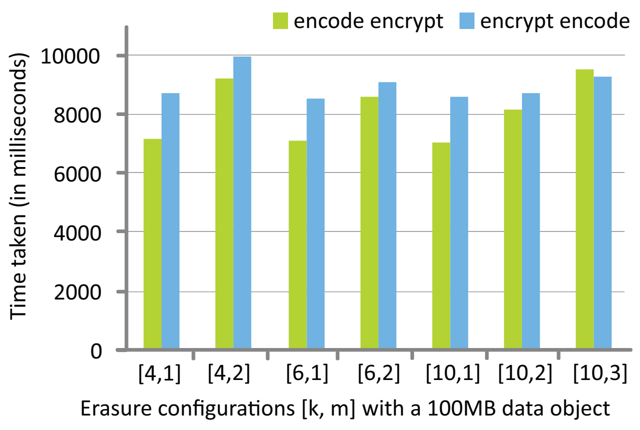
[2]The description of a code views each data package as having w bits worth of data.



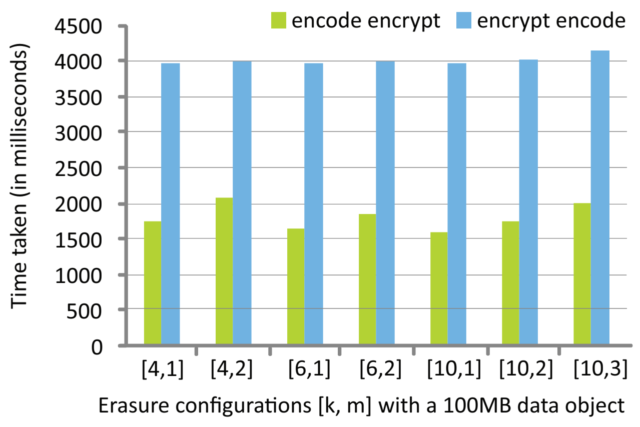
(a) Processing of a 128kB data object on a dual-core CPU



(b) Processing of a 128kB data object on a quad-core CPU



(c) Processing of a 100MB data object on a dual-core CPU



(d) Processing of a 100MB data object on a quad-core CPU

Fig. 3. Total time taken when encryption occurs either before or after the encoding step. Tests were executed on a dual-core / quad-core CPU. The bars correspond to the complete data processing cycle: the encoding of a data object into data packages, the subsequent encryption of individual chunks in parallel threads, the decryption of data packages and finally the reassembling of the data in the decoding step. The opposite order encompasses the following operations: encryption, encoding, decoding and decryption.

IV. Security

Although erasure algorithms perform a series of coding operations on data, they do not provide far reaching security functionality. There may be enough data in the encoded fragments that useful content (a username and a password or a social security number for example) could be reassembled. The only protection measure provided through erasure coding is the logical and physical segregation of the data packages, as these are distributed between different providers. Thus, we implemented a security service which enables users of our application to encrypt individual data packages prior to their transmission to cloud providers. The encryption algorithm depends on the user's security requirements specified in the user interface. In general, our implementation makes use of the AES-128 and AES-256 algorithms for data encryption. On top of this, we use SHA-1 and MD5 cryptographic hash functions to test the integrity of cloud-stored data.

A. Encryption

Concerning the security strategy, it is important to determine the point when the encryption occurs and who holds the keys to decrypt the data. In general, we performed two sets of experiments with different erasure configurations - one for initial encryption prior to the encoding step and another vise versa.

1) Experiment Setup: We employed two machines for the experiment. Neither is exceptionally high-end, but each represents a middle-range commodity processor, which should be able to encode, encrypt, decrypt and decode comfortably within the I/O speed limits of the fastest disks, where:

- Machine 1: Windows 7 Enterprise (64bit) system with an Intel Core 2 Duo E8400 @3GHz, 4 GB installed RAM and a 160 GB SATA Seagate Barracuda hard drive with 7200 U/min;
- Machine 2: Windows 7 Enterprise (64bit) system with an Intel Core i5-2500 @ 3,30GHz, 16 GB DDR3 RAM, 64bit Windows 7 Ultimate and a Seagate Barracuda hard drive with 7200 U/min.

2) Results: Figures 3 shows the results of 100 runs (per machine) executed in a random order. The test encompasses the complete data processing cycle: the encoding of a data object, its subsequent encryption, its decryption and finally the decoding step. We observe, that the processing order (encode encrypt vs. encrypt encode) does not really matter with the dual-core processor. This applies despite the fact that the usage of erasure algorithms causes an additional storage overhead (see III-C2). With regard to erasure configuration there is another factor of importance: whether the sum of the configuration attributes k and m is odd or even (see erasure configurations [4,1] and [4,2] as well as [10,1] and [10,2] in figure 3). This has an impact on the parallel processing (encryption of the data) in the following step. However, the test with a quad-core processor provides the expected results: first, the encoding of smaller data objects causes a significant higher I/O overhead and second, the encryption of larger files (executed in parallel threads) after an initial encoding step is more efficient than the opposite. With this, we made a decision to encrypt data after its being encoded into n coding packages.

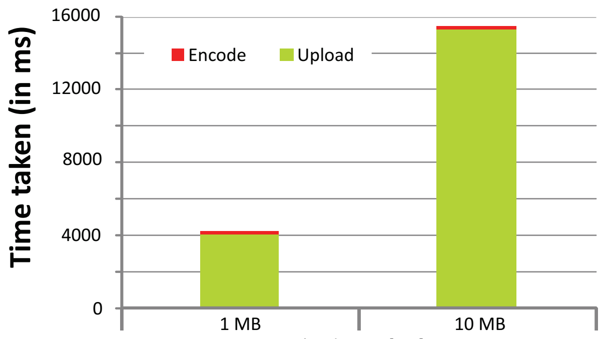
B. Key possession

Another important part when developing an encryption strategy is key the possession. The only encryption option for most of the available cloud solutions is that the keys are managed by the cloud storage providers, which is convenient to the user (the provider can assist with data restoration for example) but it entails a certain amount of risk. On one hand there are laws and policies that allow government agencies easier access to data on a cloud than on a private server. For example, in the USA the Stored Communication Act enables the FBI to access data without getting a warrant or the owner's consent. Furthermore, closed subpoenas may prohibit providers to inform their customers that data has been given to the government [19]. On another hand there is always the chance of a disgruntled employee circumventing security and using the data in a way the user never indented.

In order to provide the user 100% control over the encryption process, we store the keys locally so that no third party is able to access and read the secured data. This, however also creates a single source of failure and means that the backup of the keys and metadata required for reassembling the data is in the responsibility of the user. However, the mitigation of this issue is part of our future work and analysis.

C. Observations

To assess the impact of encryption and encoding on the overall performance of the data transmission process we performed a further experiment on our dual-core test machine. We utilized the system to transfer some data to a set of randomly selected providers. The results represented in figure 4 capture the end-to-end transmission performance of our application with files of varying sizes (1MB and 10MB). Compared with the results presented in the figure 3 we conclude that in the case of significantly higher transmission rates, encryption can be added with no noticeable performance impact.



🟥 Encode	47 ms = 0,3%	15 ms = 0,37%
🟩 Upload	15615 ms = 99,7 %	4042 ms = 99,63 %

Fig. 4. Time taken for the encoding and upload of data objects with Cloud-RAID. The encoding step requires not more than 0,5% of the entire data upload process. The data packages were sent to the following providers: Google US, Amazon EU, Amazon (US-west-1), Nirvanix, Azure and Google EU.

V. Related Work

The main idea underlying our approach is to provide RAID technique at the cloud storage level. In [1] authors introduce RACS, a proxy that spreads the storage load over several providers. This approach is similar to our work as it also employs erasure code techniques to reduce overhead while still benefiting from higher availability and durability of RAID-like systems. Our concept goes beyond a simple distribution of users' content. RACS lacks the capabilities such as intelligent file placement based on users' requirements or automatic replication. In addition to it, the RACS system does not try to solve security issues of cloud storage, but focuses more on vendor lock-in. Therefore, the system is not able to detect any data corruption or confidentiality violations.

The future of distributed computing has been a subject of interest for various researchers in recent years. In [8] Broberg and Buyya introduce a service for intelligent data placement and automatic replication which enables content creators to leverage the services of multiple cloud storage providers. However, this work does not address security and provider lock-in concerns which are mainly addressed in our approach. Further, in our work we do not aim to allocate resources from cloud providers to sell them to the customers. Our service acts as an abstraction layer between service vendors and service users automatising data placement processes. In fact, our approach enables cloud storage users to place their data on the cloud based on their security policies as well as quality of service expectations and budget preferences. Furthermore, the usage of erasure algorithms for data placement is more efficient than a native replication (in terms of storage and costs).

VI. Conclusion

In this paper we outlined some general problems of cloud computing such as security, service availability and a general risk for a customer to become dependent on a service provider. In the course of the paper we demonstrated how our system deals with the mentioned concerns. In a nutshell, we distribute users' data across multiple providers while integrating with each storage provider via appropriate service-connectors. These connectors provide an abstraction layer to hide the complexity and differences in the usage of storage services.

We use erasure code techniques for striping data across multiple providers. The first experiments proved, that given the speed of current disks and CPUs, the libraries used are fast enough to provide good performance and reliable storage system. The average performance overhead caused by data encoding is less than 0,4% of the amount of time for data transfer to a cloud provider. With this, encoding is dominated by the transmission times and can be neglected. Here, the storage overhead can be varied to achieve higher availability values depending on user requirements.

By spreading users data across multiple clouds our approach enables users to avoid the risk of data lock-in and provide a low-level protection even without using security functionality. Further, our approach has also an advantage of the user having total control over the encryption process. The first results showed, that encryption can be added with no noticeable performance impact. All in all, we enable the user to take the full responsibility for data security and redundancy depending on the individual needs and requirements. However, additional storage offerings are expected to become available in the next few years. Due to the flexible and adaptable nature of our approach, we are able to support any changes in existing storage services as well as incorporating support for new providers as they appear.

VII. FUTURE WORK

In the last month, we deployed your application using seven commercial cloud storage repositories in different countries in order to conduct a comprehensive test of our system. The results of the experiment are being analyzed currently and will be addressed in our next publication. Whilst our system is still under development at present, we will have to use the results of the conducted experiment to improve the overall performance and reliability.

REFERENCES

[1] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. Racs: A case for cloud storage diversity. *SoCC'10*, June 2010.

[2] Rehab Alnemr, Justus Bross, and Christoph Meinel. Constructing a context-aware service-oriented reputation model using attention allocation points. *Proceedings of the IEEE International Conference on Service Computing(SCC2009)*, 2009.

[3] Rehab Alnemr and Christoph Meinel. Getting more from reputation systems: A context-aware reputation framework based on trust centers and agent lists. *Computing in the Global Information Technology, International Multi-Conference*, 2008.

[4] Rehab Alnemr, Maxim Schnjakin, and Christoph Meinel. A security and high-availability layer for cloud storage. In *Web Information Systems Engineering – WISE 2010 Workshops*, volume 6724 of *Lecture Notes in Computer Science*, pages 449–462. Springer Berlin / Heidelberg, 2011.

[5] Rehab Alnemr, Maxim Schnjakin, and Christoph Meinel. Towards context-aware service-oriented semantic reputation framework. *IEEE TrustCom/IEEE ICESS/FCST, International Joint Conference of*, 0:362–372, 2011.

[6] Amazon. Amazon ec2 service level agreement. online, 2009.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. *"Technical Report UCB/EECS-2009, EECS Department, University of California, Berkeley"*, 2009.

[8] James Broberg, Rajkumar Buyya, and Zahir Tari. Creating a 'cloud storage' mashup for high performance, low cost content delivery. *Service-Oriented Computing (Volume 5472), ICSOC'08 Workshops*, April 2009.

[9] Jeffrey Burt. Future for cloud computing looks good, report says. online, 2009.

[10] Nicholas Carr. *The Big Switch*. Norton, 2008.

[11] John Gantz and David Reinsel. Extracting value from chaos. online, 2009.

[12] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[13] J. S. Plank, S. Simmerman, and C. D. Schuman. Jerasure: A library in C/C++ facilitating erasure coding for storage applications - Version 1.2. Technical Report CS-08-627, University of Tennessee, August 2008.

[14] James S. Plank. The raid-6 liberation codes. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 7:1–7:14, Berkeley, CA, USA, 2008. USENIX Association.

[15] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance free global storage in oceanstore. *IEEE Internet Computing*, September 2001.

[16] Maxim Schnjakin, Rehab Alnemr, and Christoph Meinel. Contract-based cloud architecture. In *Proceedings of the second international workshop on Cloud data management*, CloudDB '10, pages 33–40, New York, NY, USA, 2010. ACM.

[17] Maxim Schnjakin and Christoph Meinel. Platform for a secure storage-infrastructure in the cloud. *Proceedings of the 12th Deutscher IT-Sicherheitskongress (Sicherheit 2011)*, 2011.

[18] The Amazon S3 Team. Amazon s3 availability event: July 20, 2008. online, 2008.

[19] Anthony T.Velte, Toby J. Velte, and Robert Elsenpeter. *Cloud Computing: A Practical Approach*. Mc Graw Hill, 2009.