Chapter 15
Solving Security and Availability Challenges in Public Clouds

Maxim Schnjakin
Potsdam University, Germany

Christoph Meinel
Potsdam University, Germany

ABSTRACT

Cloud Computing as a service-on-demand architecture has grown in importance over the previous few years. One driver of its growth is the ever-increasing amount of data that is supposed to outpace the growth of storage capacity. The usage of cloud technology enables 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 chapter, the authors present a system that improves availability, confidentiality, and reliability of data stored in the cloud. To achieve this objective, the authors encrypt users' data and make use of the RAID-technology principle to manage data distribution across cloud storage providers. Further, they discuss the security functionality and present a proof-of-concept experiment for the application to evaluate the performance and cost effectiveness of the approach. The authors deploy the application using eight commercial cloud storage repositories in different countries. The approach allows users to avoid vendor lock-in and reduces significantly the cost of switching providers. They also observe that the implementation improved the perceived availability and, in most cases, the overall performance when compared with individual cloud providers. Moreover, the authors estimate the monetary costs to be competitive to the cost of using a single cloud provider.

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 (Carr, 2008), (Armbrust et al., 2010).

DOI: 10.4018/978-1-4666-6158-5.ch015

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 (Burt, 2009) 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 (Gantz, & Reinsel, 2009), 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 solely on 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 (Ponemon Institute, 2011). 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 the physical control 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 distributing corporate data among multiple storage providers. Such an approach is pursued by content delivery networks (for example in (Broberg, Buyya, & Tari, 2009), (Buyya, Yeo, & Venugopal, 2008) and 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 done for years at the level of file systems and disks. In our work we use RAID like 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 problem by encrypting and encoding 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 (bank-

ruptcy), software or network failure, etc. In July 2008, for instance, Amazon storage service S3 was down for 8 hours because of a single bit error (The Amazon S3 Team., 2008). 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 recent times, no spectacular outages were observed. Nevertheless failures do occur. For example, in October 2009 a subsidiary of Microsoft, Danger Inc., lost the contracts, notes, photos, etc. of a large number of users of the Sidekick service (Sarno, 2009). 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 his 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. In recent months we conducted an extensive experiment for our application to evaluate the overall performance and cost effectiveness of the approach. In the current work we present the design of our application and the results of the experimental study. We show, that with an appropriate coding configuration Cloud-RAID is able to improve significantly the performance of the data transmission process, whereby the monetary costs are competitive to the cost of using a single cloud.

ARCHITECTURE

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, supporting 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 (Redundant Arrays of Inexpensive Disks). 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 chapter IV). This enables us to tolerate the loss of one or more storage providers without suffering any loss of content (Weatherspoon & Kubiatowicz 2002), (Dingledine, Freedman, & Molnar, 2000).

The system has a number of core components that contain the logic and management layers required to encapsulate the functionality of different storage providers. 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). User can upload, view, modify or delete existing content. Further, the user is presented with options to specify parameters regarding security or storage and transfer budget.
- **Resource Management Module:** This system component is responsible for intelligent deployment of data based on users' requirements. The component is supported by:
  - A registry and matching service: assigns storage repositories based on users requirements (for example physical location of the service, costs and performance expectations). Monitors the performance of participating providers and ensures that they are meeting the agreed SLAs.
  - A resource management service: takes operational decisions regarding the content storage.
  - A task scheduler service: has the ability to schedule the launch of operations at peak-off hours or after specified time intervals.
- **Data Management Module:** This component handles data management on behalf of the resource management module and is mainly supported by:
  - A data encoding service: this component is responsible for striping and encoding of user content.
  - A data distribution service: spreads the encoded data packages across multiple providers. Since each storage service is only accessible through a unique API, the service utilizes storage "service connectors", which provide an abstraction layer for the communication to storage repositories.
  - A security service: manages the security functionality based on a user's requirements (encryption, secret key management).

Further details can be found in our previous work (Schnjakin, & Meinel, 2011), (Schnjakin, Alnemr, & Meinel, 2010), and (Schnjakin, Alnemr, & Meinel, 2011).

DESIGN

Any application needs a model of storage, a model of computation and a model of communication. 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 (Plank, Simmerman, & Schuman. 2008) 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.

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 (Schnjakin, Alnemr, & Meinel, 2010) 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, for example:

- Budget-oriented content deployment (based on the price model of available providers).
- Data placement based on quality of service parameters (for example availability, throughput or average response time).
- Storage of data based on geographical regions of the user's choice. The restriction of data storage to specific geographic areas can be reasonable in the case of legal restrictions.

Storage Repositories

Cloud Storage Providers

Cloud storage providers are modeled as a storage entity that supports six basic operations, shown in Table 1. We need storage services to support not more than the aforementioned operations.

Further, the individual providers are not trusted. This means that the entrusted data can be corrupted, deleted or leaked to unauthorized parties. This fault model encompasses both malicious attacks on a provider and arbitrary data corruption like the Sidekick case (section 1). 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 and Nirvanix SND. Further providers can be easily added.

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. Until

Table 1. Storage connector functions

Function	Description
create(ContainerName)	creates a container for a new user
write(ContainerName, ObjectName)	writes a data object to a user container
read(ContainerName, ObjectName)	reads the specified data object
list(ContainerName)	list all data objects of the container
delete(ContainerName, ObjectName)	removes the data object from the container
getDigest(ContainerName, ObjectName)	returns the hash value of the specified data object

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 (Schnjakin, Alnemr, & Meinel, 2010). 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.

Matching

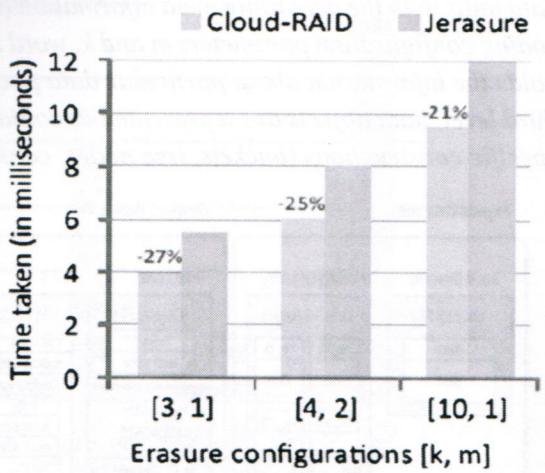
The selection of storage services for the data distribution occurs based on user preferences set in the user interface. After matching user requirements and provider capabilities, we use the reputation of the providers to produce the final list of potential providers to host parts of the user's data. 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). 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.

Data Management

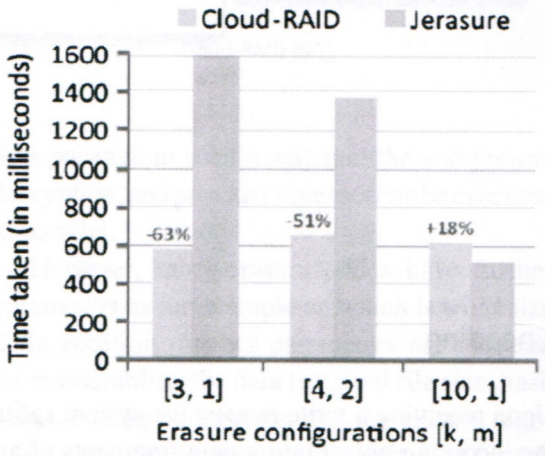
Data Model

In compliance with (Abu-Libdeh, Princehouse, & Weatherspoon, 2010), we mimic the data model of Amazon's S3 by the implementation of our encoding and distribution service. All data objects

Figure 1. Total time taken when Jerasure and Cloud-RAID libraries are used to encode data objects of varying sizes



(a) Encoding of a 100kB data object

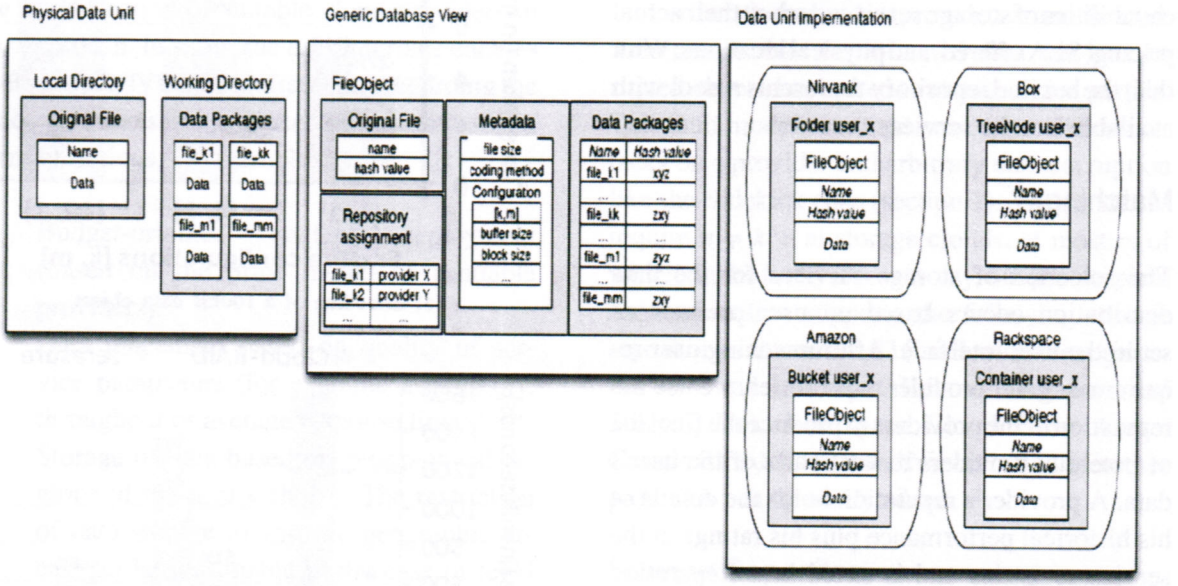


(b) Encoding of a 100MB data object

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 (see Figure 2).

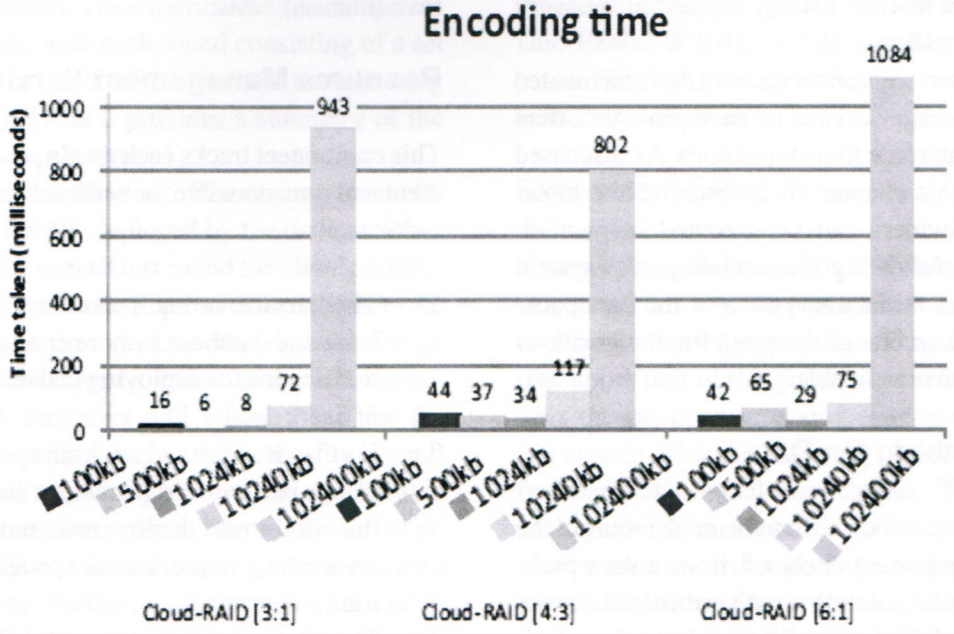
Figure 2. Data unit model at different abstraction levels. At a physical layer (local directory) each data unit has a name (original file name) and the encoded k+m data packages. In the second level, Cloud-RAID perceives data objects as generic data units in abstract clouds. Data objects are represented as data units with the according meta information (original file name, cryptographic hash value, size, used coding configuration parameters m and k, word size etc.). The database table "Repository Assignment" holds the information about particular data packages and their (physical) location in the cloud. In the third level, data objects are represented as containers in the cloud. Cloud-RAID supports various cloud specific constructions (buckets, tree nodes, containers etc.).



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 (Plank, Simmerman, & Schuman. 2008). 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 (Rhea et al., 2001). The total size of all data packets (after encoding) can be expressed with the following equation: $(s/k * k)+(s/k * m) = s + (s/k * m) = s * (1 + m/k)$. With this, the usage of erasure codes increases the total storage by a factor of m/k. Summarized, the overall overhead depends on the file size and the defined m and k parameters for the erasure configuration. Figure 3 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 (Plank, Simmerman, & Schuman. 2008) 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 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 (Plank, 2008) for example is a

Figure 3. The average performance of the erasure algorithm with data objects of varying sizes (100kB, 500kB, 1MB, 10MB and 100MB)



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 (Plank, Simmerman, & Schuman. 2008) 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 1 summarizes the results of 20 runs executed on test machine 1.

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 backend database after performing a successful write request.

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 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. Therefore, following the encoding, the system performs an initial encryption of the data packages based on one of the predefined algorithms (this feature is optional).

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. A read request can be directed to a random data share or the physically closest service (latency optimal approach).

Another possible approach is to fetch data from service providers that meet certain performance criteria (e.g response time or throughput). Finally, there is a minimal-cost aware policy, which guides user requests to the cheapest sources (cost optimal approach). The latter strategy is implemented as a default configuration in our system. Other more sophisticated features as a mix of several complex criteria (e.g. faults and overall performance history) are under development at present. However, the read optimization has been implemented to save time and costs.

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 (Schnjakin, Alnemr, & Meinel, 2011).

PERFORMANCE EVALUATION

In this section we present an evaluation of our system that aims to clarify the main questions concerning the cost, performance and availability aspects when erasure codes are used to store data on public clouds.

Methodology

The experiment was run on Hasso Plattner Institute (HPI), which is located close to Berlin, Germany, over a period of over 377 (24x7) hours, in the middle of July 2012. As it spans seven days, localized peak times (time-of-day) is experienced in each geographical region. HPI has a high speed connectivity to an Internet backbone (1 Gb), which ensures that our test system is not

a bottleneck during the testing. The global testbed spans eight cloud providers in five countries on three continents. The experiment time comprises three rounds, with each round consisting of a set of predefined test configurations (in the following sequences). Table 2 provides a summary of the conducted experiment.

We used test files of different sizes from 100 kB up to 100MB, deployed by the dedicated test clients. Prior to each test round the client requires a persistent connection to the APIs of the relevant cloud storage providers, so that requests for an upload or download of test data can be send. In general, providers will refuse a call for the establishment of a new connection after several back-to-back requests. Therefore we implemented an API connection holder. After two hours of an active connection the old connection is overwritten by a new one. Further, we determine a timeout of one second between two unsuccessful requests, each client waits for a think time before the next request is generated.

Machines for Experimentation

We employed three machines for experimentation. Neither is exceptionally high-end, but each represents 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. These are: 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.

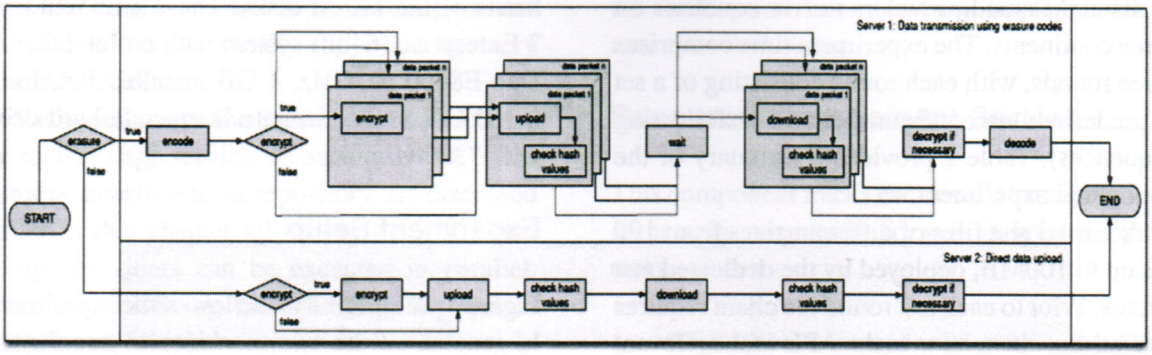
Experiment Setup

Figure 4 presents the workflow of the experiment. In general we use two machines to transfer test data to cloud storage providers. The first machine (the upper part of the graph) uses erasure codes. This means, upon receiving a write request the test system splits the incoming object into k data fragments of an equal size - chunks . These *k* data packages hold the original data. In the next step the system adds *m* further packages whose contents are calculated from the *k* chunks, whereby *k* and *m* are variable parameters (Plank, Simmerman, & Schuman. 2008). With this, 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* shares is sufficient to reconstruct the original data object (Rhea et al., 2001).

Table 2. Experiment details

Category	Description
Cloud storage provider	8
Locations	Europe, USA, Asia
Total experiment time	about 15d 9h (377h)
Total number of test rounds	about 3 rounds
Total number of requests (read/write) / round	281,900
Service time out for each request	1 sec
Test file size	100 kB - 100 MB
Coding Method	cauchy_good
Coding configuration [k,m]	k=[2..4,6,10], m=[1..2], k>=m

Figure 4. Workflow of the experiment



In the next step, the application makes sure that each 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 encoding, encryption, decryption, and provider accesses can be executed in parallel. The second machine (the lower part of the graph in the Figure 4) uploads the entire data object to a single provider without any modifications. As we are interested in the direct comparison between these two approaches, we want each data transmission to start simultaneously.

Therefore we used the third machine as a "sync-instance" running a Tomcat 7 server with a self-written sync-servlet which controls the workflow of the experiment.

Erasure Configuration

In our experiment we make use of the Cauchy-Reed-Solomon algorithm for two reasons. First, according to Plank et al. (Plank et al., 2009a) 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 various coding techniques which are available to the public. Second, the algorithm allows free selection of coding parameters k and m, whereas other algorithms restrict the choice of parameters. Liberation Code (Plank, 2008) for example is a specification for storage systems with $n = k + 2$ nodes to tolerate the failure of any two nodes (whereby the parameter m is fix and is equal to two).

In our test scenario we tested more than 2520 combinations of k and m. We will denote them by $[k,m]$ in the course of the chapter, whereby the present evaluation focuses on an encoding configuration [4,1]. Which means, that the setting provides data availability toward one cloud failure at the time of read or write request. Most of the providers have SLAs with 99% and 99.9% monthly up-time percentages. Thus, we believe that adding enough redundancy to tolerate provider outage or failure at a time will be sufficient in most cases.

Schemes and Metrics

The goal of our test is to evaluate the performance of our approach. Mainly we are interested in availability of APIs, overhead caused by erasure codes and transmission rates. Therefore, we implemented a simple logger application to record the results of our measurements. In total we log 34 different events. For example, each state of the workflow depicted in Figure 4 is captured with two log entries (START and END).

Erasure Overhead

Due to the nature of erasure codes, each file upload and download is associated with a certain overhead. As discussed in before the total size of all chunks (after encoding) can be expressed with the

following equation: $s+(s/k * m) = s * (1 + m/k)$, whereby variable s is defined as the original file size. Again, the usage of erasure codes increases the total storage by a factor of m/k. Further, we need to encode data prior to its upload and accordingly decode the downloaded packets into the original file. Both operations cause an additional computational expense.

Transmission Performance and Throughput

We measure the throughput obtained from each read and write request. In general the throughput is defined as the average rate of successful message delivery over a communication channel. In our work we link the success of the message delivery to the success of the delivery of the entire data object. In our approach, a data object is completely transferred, when the last data package is being successfully transferred to the transfer destination. This means that in case of data upload, the transfer is only completed, when (upon a write request) our client receives a confirmation message in the form of individual digest values that correspond with the results of the local computation (this applies for all transferred data packages). In the event of a mismatch the system will delete the corrupted data and initiate a re-upload procedure. With this, the value of throughput does not only represent the pure upload or download rate of the particular providers, as the measured time span includes also possible failures, latency and the bilateral processing of get-hash calls.

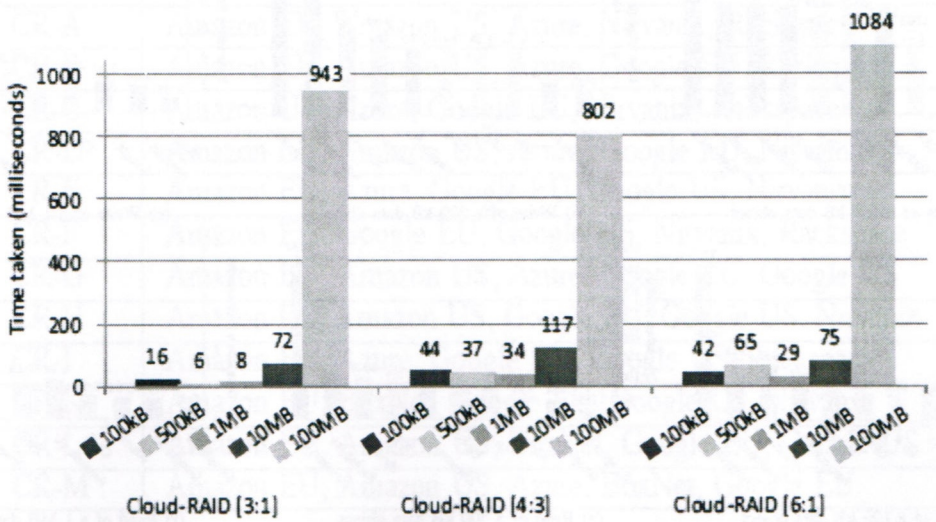
Empirical Results

This section presents the results in terms of read and write performance, as well as throughput, response time and availability based on over 281.000 requests. Due to space constraints, we present only some selected results from the conducted experiment.

Erasure Overhead

As described in IV-B1 the erasure coding leads to a storage overhead of factor m/k. For instance, an $[k=4, m=1]$ encoding results in a storage overhead of $14\ 100\% = 25\%$. In order to reduce the storage

Figure 5. The computational overhead caused by erasure with different configurations and file sizes. In general, the overall overhead increases with growing file size regardless of the defined m and k parameters for the erasure configuration.



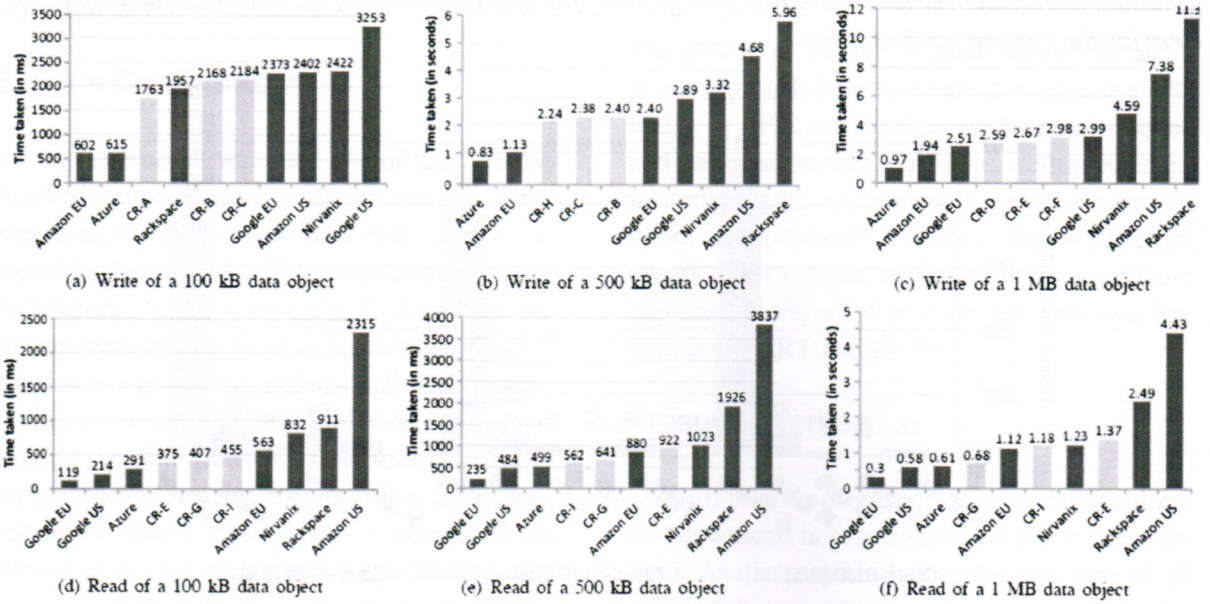
overhead, it would be advisable to define high k and preferably low m values. For example, an encoding configuration *[k = 10, m = 1]* produces a storage overhead of ¼ * 100% = 25%. In order to reduce the storage overhead, it would be advisable to define high *k* and preferably low *m* values. For example, an encoding configuration *[k=10, m=1]* produces 1/10 * 100% = 10%. Erasure causes also a computational overhead. During the experiment we scrutinized 12 different configurations. A selection of the results is presented in Figure 5. The figure illustrates, that the computational expense increases with the file size regardless of the erasure configuration. As the encoding of a 100 MB data object takes approximately one second, the encoding overhead can be neglected in view of the significantly higher transmission times. In (Schnjakin et al., 2013) we showed, that the average performance overhead caused by data encoding is less than 2% of the entire data transfer process to a cloud provider.

Using encryption, we can say that the total performance decreases as individual data packages have to be encrypted locally before moving them to the cloud. In our experiments the costs for encryption were less than 3% of total time which is also negligible in view of the overall transmission performance. This point has been addressed in our previous work (Schnjakin et al., 2013) and (Schnjakin et al., 2013).

Transmission Performance and Throughput

Due to space constraints the current evaluation focuses on the Cloud-RAID configuration with k = 4 and m = 1. For performance comparison we experimented with different combinations among eight clouds, which are: Amazon US, Amazon EU, Azure, Box, Google EU, Google US, Nirvanix and Rackspace. The particular combinations are represented in Table 3.

Figure 6. Average throughput performance in milliseconds and seconds observed on all reads and writes executed for the [4,1] Cloud-RAID configuration (4 of 5 data packages are necessary to reconstruct the original data, m = 1). The Cloud-RAID bars (CR) correspond to the complete data processing cycle: the encoding of a data object into data packages and the subsequent transmission of individual chunks in parallel threads.



(a) Write of a 100 kB data object

(b) Write of a 500 kB data object

(c) Write of a 1 MB data object

(d) Read of a 100 kB data object

(e) Read of a 500 kB data object

(f) Read of a 1 MB data object

In general, we observed that utilizing Cloud-RAID for data transfer improves the throughput significantly when compared with cloud storages individually. This can be explained with the fact, that Cloud-RAID reads and writes a fraction of the original data (more specific 14 th with [4,1] setting, see IV-B1) from and to clouds simultaneously.

However, the total time of data transfer depends on the throughput performance of each provider involved into the communication process. The throughput performance of Cloud-RAID increases with higher performance values of cloud providers involved into the data distribution setting.

During the performance evaluation we observed, that storage providers differ extremely in their upload and download capabilities. Moreover, some vendors seem to have optimized their infrastructure for large files, while others focused more on smaller data objects. In the following we will clarify this point.

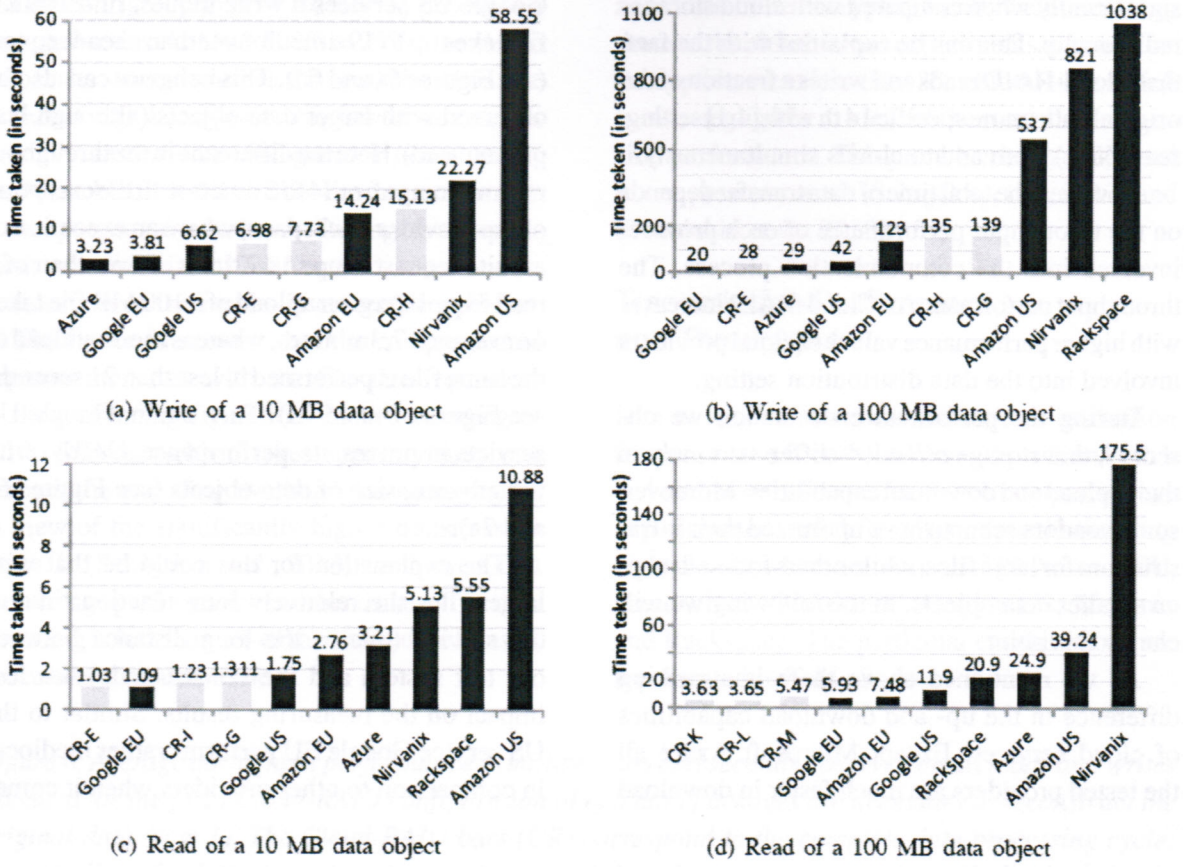
As we mentioned above there is a striking difference in the up- and download capabilities of cloud services. Except Microsoft Azure all the tested providers are much faster in download than in upload. This applies to smaller and larger data objects. At one extreme, with Google EU or Google US services a write request of a 100 kB file takes up to 19 times longer than a read request (see Figures 6a and 6d). This behavior can also be observed with larger data objects (although less pronounced). Here the difference in the throughput rate may range from 4 to 5 times, with the exception of the provider Rackspace, where an execution of a write request is up to 49 times slower than of a read request (e.g. an upload of a 100 MB file takes on average 17,3minutes, whereas the download of the same file is performed in less than 21 seconds, see Figures 7b and 7d). Then again, Google US service improves its performance clearly with the growing size of data objects (see Figures 6a and 7a).

The explanation for this could be that with larger files the relatively long reaction time of the service (due to the long distance between our test system and the service node) has less impact on the measuring results. Similar to the US service Google EU performs rather mediocre in comparison to other providers when it comes

Table 3. Cloud-RAID setting with k = 4 AND m = 1

Cloud-RAID	Provider Setting
CR-A	Amazon EU, Amazon US, Azure, Nirvanix, Rackspace
CR-B	Amazon EU, Amazon US, Azure, Google EU, Rackspace
CR-C	Amazon US, Azure, Google EU, Nirvanix, Rackspace
CR-D	Amazon EU, Amazon US, Azure, Google EU, Nirvanix
CR-E	Amazon EU, Azure, Google EU, Google US, Nirvanix
CR-F	Amazon EU, Google EU, Google US, Nirvanix, Rackspace
CR-G	Amazon EU, Amazon US, Azure, Google EU, Google US
CR-H	Amazon EU, Amazon US, Google EU, Google US, Nirvanix
CR-I	Amazon EU, Azure, Google EU, Google US, Rackspace
CR-K	Amazon EU, BoxNet, Google EU, Google US, Nirvanix
CR-L	Amazon EU, Amazon US, BoxNet, Google EU, Google US
CR-M	Amazon EU, Amazon US, Azure, BoxNet, Google EU

Figure 7. Throughput observed in seconds on reads and writes executed for the [4,1] Cloud-RAID configuration. Here again, CR bars correspond to the complete data processing cycle.



(a) Write of a 10 MB data object

(b) Write of a 100 MB data object

(c) Read of a 10 MB data object

(d) Read of a 100 MB data object

to read speeds for data objects up to 1 MB, (see Figures 6a and 6b). In terms of performance for writing larger files,

Google EU becomes the clear leader and even outperforms the fastest Cloud-RAID setting, which consists of the five fastest providers: Amazon EU, Azure, Google EU, Google US and Nirvanix (see Figure 7b). Similar phenomena have been observed by read requests. Microsoft Azure belongs to the leading providers for reading 100 kB data objects (see Figure 6d) and falls back by reading 100 MB files (see Figure 7d).

Hence, the performance of Cloud-RAID differs depending on the provider setting and file size. It is observed that our systems achieves better throughput values for read requests. The reason is that the test client fetches less data from the cloud (only k of n data packages) than in case of a write request, where all n packages have to be moved to the cloud.

As expected, we observe that the fastest read and write settings consist of the fastest clouds. Concerning writing 100 kB data objects, the fastest Cloud-RAID setting CR A improves the overall throughput by an average factor of 3 (compared to the average throughput performance of the providers in the current Cloud-RAID setting). For reading 100 kB, CR-E achieves an improvement factor of 5. In terms of performance for writing 1 MB and 10 MB objects, Cloud-RAID setting CR-D and CR-E achieve already an average improvement factor of 7. Then again, for reading 10 MB, Cloud-RAID improves the average performance by a factor of 13 and even outperforms the fast-

est cloud providers (see figure 7c). By smaller data objects, execution of both read and write requests is highly affected by erasure overhead, DNS lookup and API connection establishment time. This can lead to an unusual behavior. For example, the transmission of a 100 kB data object to Google US can take our system more time than the transmission of a 500 kB or even 1 MB file (see Figure 6a, 6b and 6c). Hence, increasing the size of data objects improves the overall throughput of Cloud-RAID. Concerning read and write speeds for 100 MB data objects, Cloud-RAID increases the average performance by a factor of 36 for writes (despite of the erasure overhead of 25 percent) and achieves an improvement factor of 55 for reads (see Figures 7c and 7d).

There is also an observed connection between the throughput rate and the size of data objects. Charts 6a to 6f show results from performance tests on smaller files (up to 1 MB). Microsoft Azure and Amazon EU achieve the best results in terms of write requests. When writing 10 MB or 100MB data objects Amazon EU falls back on the fourth place (see Figures 7b and 7d). Form these observations, we come to the following conclusions. The overall performance of Cloud-RAID is not only dependent on the selection of k and m values, but also on the throughput performance of the particular storage providers. Cloud-RAID increases the overall transmission performance compared to the slower providers. Beyond that we are able to estimate, that the more providers are involved into the data distribution process, the less weight slower providers carry in terms of overall throughput performance. The underlying reason is again the size of individual data packages, which decrease with the growing number of *k* data packages (see chapter IV-B1).

Observations and Economic Consequences

Finally, based on the measured observations, we determine users benefits from using our system. In order to assert the feasibility of our application we have to examine the cost structure of cloud storage services. Vendors differ in pricing scheme and performance characteristics. Some providers charge a flat monthly fee, others negotiate contracts with individual clients. However, in general pricing depends on the amount of data stored and bandwidth consumed in transfers. Higher consumption results in increased costs.

As illustrated in Tables 4 and 5 providers also charge per API request (such as read, write, get-hash, list etc.) in addition to bandwidth and storage. The usage of erasure codes increases the total number of such requests, as we divide each data object into chunks and stripe them over multiple cloud vendors. The upload and download of data takes on average two requests. Considering this, our system needs $(4+1)2 = 10$ requests for a single data upload with a [4, 1] coding configuration. The download requires only 4 2 = 8 requests, as merely 4 packets have to be received to rebuild the original data. Thus, erasure [k,m] increases the number of requests by a factor of k + m for upload and k for download.

Consequently, the usage of erasure codes increases the total cost compared to a direct upload or download of data due to the caused storage and API request overhead. Tables 4 and 5 summarize the cost in US Dollars of executing 10,000 reads and 10,000 writes with our system considering 5 data unit sizes: 100 kB, 500 kB, 1 MB, 10 MB and 100 MB. We observe, that the usage of erasure is not significantly more expensive than using a single provider. In some cases the costs can be even reduced.

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

Table 4. Costs in dollars for 10,000 reads

Provider	Filesize in kB				
	100	500	1024	10240	102400
CR-B	0.15	0.55	1.07	10.21	101.61
CR-G	0.16	0.52	0.99	9.28	92.25
CR-I	0.15	0.55	1.07	10.21	101.61
CR [6,1][1]	3.61	4.12	4.78	16.50	133.69
Azure	0.11	0.53	1.08	10.74	107.42
Amazon/Google	0.13	0.59	1.19	11.74	117.21
Rackspace	0.17	0.86	1.76	17.58	175.78
Nirvanix	4.14	4.72	5.46	18.65	150.48

[1] The setting CR [6,1] consist of nearly all providers involved in the test setting: Amazon EU, Amazon US, Azure, Boxnet, Google EU, Nirvanix, Rackspace.

Table 5. Costs in dollars for 10,000 writes

Provider	Filesize in kB				
	100	500	1024	10240	102400
CR-B	0.12	0.12	0.12	0.12	0.12
CR-G	0.16	0.16	0.16	0.16	0.16
CR-I	0.12	0.12	0.12	0.12	0.12
CR [6,1]	8.14	8.20	8.29	9.75	24.40
Azure	0.00	0.00	0.00	0.00	0.00
Amazon/Google	0.02	0.02	0.02	0.02	0.02
Rackspace	0.00	0.00	0.00	0.00	0.00
Nirvanix	4.10	4.48	4.98	13.77	101.66

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.

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.

Figures 9 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. 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 9). 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.

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. 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.

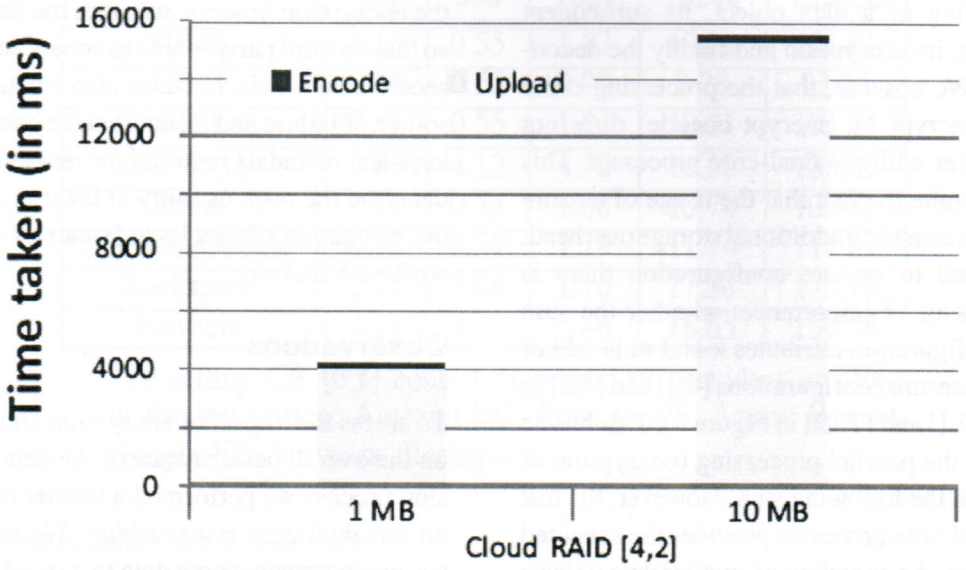
Observations

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 8 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 9 we conclude that in the case of significantly higher transmission rates, encryption can be added with no noticeable performance impact.

RELATED WORK

The main idea underlying our approach is to provide RAID technique at the cloud storage level. In (Bowers et al., 2009) the authors introduce the HAIL (High-Availability Integrity Layer) system, which utilizes RAID-like methods to manage remote file integrity and availability across a collection of servers or independent storage services. The system makes use of challenge-response protocols for retrievability (POR) (Ateniese et al., 2007) and proofs of data possession (PDP) (Ateniese et al., 2007) and unifies these two approaches. In comparison to our work, HAIL requires storage providers to run some code whereas our system

Figure 8. 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.



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

deals with cloud storage repositories as they are. Further, HAIL does not provide confidentiality guarantees for stored data.

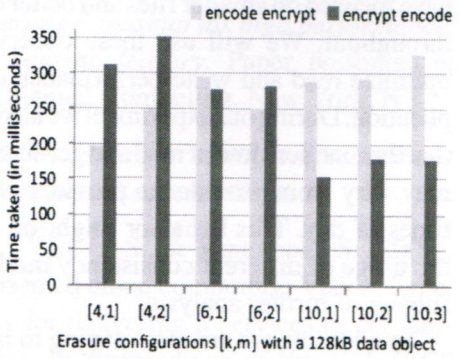
In (Dabek et al., 2001) Dabek et al. use RAID-like techniques to ensure the availability and durability of data in distributed systems. In contrast to the mentioned approaches our system focuses on the economic problems of cloud computing described in chapter I. Further, in (Abu-Libdeh, Princehouse, & Weatherspoon, 2010) 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 sophisticated 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.

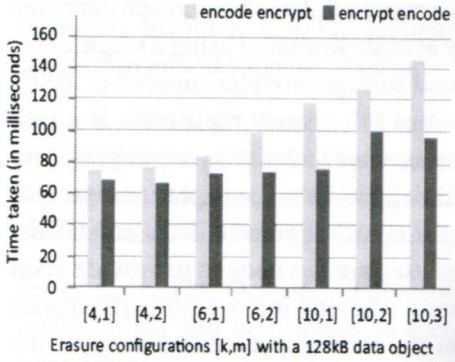
The future of distributed computing has been a subject of interest for various researchers in recent years. The authors in (Buyya, Yeo, & Venugopal, 2008) propose an architecture for market-oriented allocation of resources within clouds. They discuss some existing cloud platforms from the market-oriented perspective and present a vision for creating a global cloud exchange for trading services. The authors consider cloud storage as a low-cost alternative to dedicated Content Delivery Networks (CNDs).

There are more similar approaches dealing with high availability of data trough its distribution among several cloud providers. DepSky-A (Bessani et al., 2011) protocol improves availability and integrity of cloud-stored data by replicating

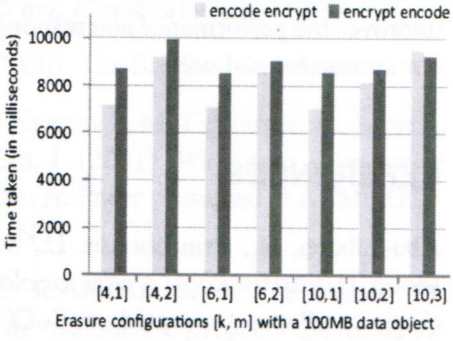
Figure 9. Total time taken when encryption occurs either before or after the encoding step. Tests were executed on a dual-core/quad-core CPU. The bars correspond to the complete data processing cycle: the encoding of a data object into data packages, the subsequent encryption of individual chunks in 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.



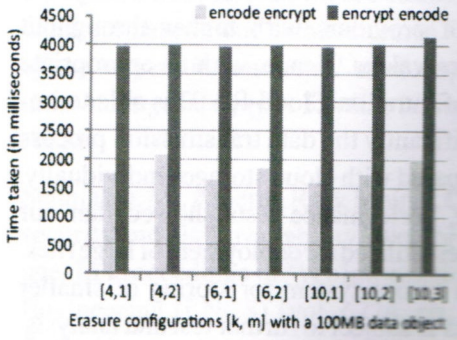
(a) Processing of a 128kB data object on a dual-core CPU



(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

it on cloud providers using quorum techniques. This work has two main limitations. First, a data unit of size S consumes *n x S* storage capacity of the system and costs on average n times more than if was stored on a single cloud. Second, the protocol does not provide any confidentiality guaranties, as it stores the data in clear text. In their later work the authors present DepSky-CA, which solves the mentioned problems by the encryption of the data and optimization of the write and read process. However, the monetary costs of using the system is still twice the cost of using a single cloud. On top of this, DepSky does not provide any means or metrics for user centric data placement. 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.

CONCLUSION

In this chapter 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 stripe 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.

The main focus of the paper is an extensive evaluation of our application. From the results obtained, we conclude that our approach improves availability at costs similar to using a single commercial cloud storage provider (instead of 100% and more when full content replication is used). We use erasure code techniques for striping data across multiple providers. The experiment proved, that given the speed of current disks and CPUs, the libraries used are fast enough to provide good performance - whereby the overall performance depends on the throughput performance of the particular storage providers. The throughput performance of Cloud-RAID increases with the selection of providers with higher throughput performance values. Hence, with an appropriate coding configuration Cloud-RAID is able to improve significantly the data transmission process when compared with cloud storages individually.

Further, performance tests showed that our system is best utilized for deployment of large files. Utilization of our system for storing of smaller data objects is subject to further test and analysis. In the long term, our approach might foster the provision of new and even more favorable cloud storage services. Today, storage providers surely use RAID like methods to increase the reliability of the entrusted data to their customers. The procedure causes costs which are covered by providers price structure. With our approach, the on-site backups might become redundant, as users data is distributed among dozens of storage services. Furthermore, we enable users of cloud storage services to control the availability and physical segregation of the data by themselves. 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.

FUTURE WORK

Our performance testing revealed that some vendors have optimized their systems for large data objects and high upload performance, while others have focused on smaller files and better download throughput. We will use these observations to optimize read and write performance of our application. During our experiment we also observed that the reaction time of read and get-hash requests may vary from provider to provider at different times of day. This behavior might be related to the usage of different consistency models and is subject of further analysis.

In addition, we are also planing to implement more service connectors and thus to integrate additional storage services. Any extra storage resource improves the performance and responsiveness of our system for end-users.

REFERENCES

Abu-Libdeh, H., Princehouse, L., & Weatherspoon, H. (2010). *Racs: A case for cloud storage diversity*. Paper presented at SoCC'10. New York, NY.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., & Konwinski, A. et al. (2010, April). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58. doi:10.1145/1721654.1721672

Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., & Song, D. (2007). *Provable data possession at untrusted stores*. Paper presented at the 14th ACM CCS. New York, NY.

Bessani, A., Correia, M., Quaresma, B., Andre, F., & Sousa, P. (2011). Depsky: Dependable and secure storage in a cloud-of-clouds. In *Proceedings of the Sixth Conference on Computer Systems*, (pp. 31–46). New York, NY: ACM.

Bowers, K. D., Juels, A., & Oprea, A. (2009). *Hail: A high availability and integrity layer for cloud storage.* Paper presented at CCS'09. New York, NY.

Broberg, J., Buyya, R., & Tari, Z. (2009). *Creating a 'cloud storage' mashup for high performance, low cost content delivery.* Paper presented at Service-Oriented Computing. New York, NY.

Burt, J. (2009). *Future for cloud computing looks good, report says.* Academic Press.

Buyya, R., Yeo, C.-S., & Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications.* IEEE.

Carr, N. (2008). The Big Switch. Norton.

Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., & Stoica, I. (2001). *Wide-area cooperative storage with cfs.* Paper presented at ACM SOSP. New York, NY.

Dingledine, R., Freedman, M., & Molnar, D. (2000). *The freehaven project: Distributed anonymous storage service.* Paper presented at the Workshop on Design Issues in Anonymity and Unobservability. New York, NY.

Gantz, J., & Reinsel, D. (2009). *Extracting value from chaos.* Academic Press.

Plank, J. S. (2008). The raid-6 liberation codes. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies.* Berkeley, CA: USENIX Association.

Plank, J. S., Luo, J., Schuman, C. D., Xu, L., & Wilcox-O'Hearn, Z. (2009a). A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proceedings of the 7th conference on File and storage technologies,* (pp. 253–265). Berkeley, CA: USENIX Association.

Plank, J. S., Simmerman, S., & Schuman, C. D. (2008). *Jerasure: A library in C/C++ facilitating erasure coding for storageapplications - Version 1.2* (Technical Report CS-08-627). University of Tennessee.

Ponemon Institute. (2011). *Security of cloud computing providers study.* Author.

Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Weatherspoon, H., & Kubiatowicz, J. (2001, September). Maintenance free global storage in ocean store. *IEEE Internet Computing.*

Sarno, D. (2009, October). Microsoft says lost sidekick data will be restored to users. *Los Angeles Times.*

Schnjakin, M., Alnemr, R., & Meinel, C. (2010). Contract-based cloud architecture. In *Proceedings of the second international workshop on Cloud data management,* (pp. 33–40). New York, NY: ACM.

Schnjakin, M., Alnemr, R., & Meinel, C. (2011). A security and high-availability layer for cloud storage. In Proceedings of WebInformation Systems Engineering (LNCS) (vol. 6724, pp. 449–462). Springer.

Schnjakin, M., Goderbauer, M., Krueger, M., & Meinel, C. (2013). Cloud storage and it-security. In *Proceedings of the 13th Deutscher IT-Sicherheitskongress* (Sicherheit 2013). Academic Press.

Schnjakin, M., Korsch, D., Schoenberg, M., & Meinel, C. (2013). Implementation of a secure and reliable storage above the untrusted clouds. In *Proceedings of Computer Science & Education (ICCSE),* (pp. 347-353). ICCSE.

Schnjakin, M. & Meinel, C. (2011). Platform for a secure storage-infrastructure in the cloud. In *Proceedings ofthe 12th Deutscher IT-Sicherheitskongress* (Sicherheit 2011). Academic Press.

The Amazon S3 Team. (2008). *Amazon s3 availability event: July 20, 2008.* Author.

Weatherspoon, H. & Kubiatowicz, J. (2002). *Erasure coding vs. replication: A quantitative comparison*. IPTPS.

ADDITIONAL READING

Dimitrios Zissis and Dimitrios Lekkas. (2012). Addressing cloud computing security issues. *Future Gener. Comput. Syst.* 28, 3 (March 2012), 583-592. DOI=10.1016/j.future.2010.12.006

Krutz, R. L., & Vines, R. D. (2010). *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. Wiley Publishing.

Madhan Kumar Srinivasan, K. Sarukesi, Paul Rodrigues, M. Sai Manoj, and P. Revathy. (2012). State-of-the-art cloud computing security taxonomies: a classification of security challenges in the present cloud computing environment. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics* (ICACCI '12). ACM, New York, NY, USA, 470-476. DOI= doi:10.1145/2345396.2345474 http://doi.acm.org/10.1145/2345396.2345474

Perez-Botero, D., Szefer, J., & Lee, R. B. (2013). Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 international workshop on Security in cloud computing* (Cloud Computing '13). ACM, New York, NY, USA, 3-10.

S. Subashini and V. Kavitha. (2011). Review: A survey on security issues in service delivery models of cloud computing. *J. Netw. Comput. Appl.* 34, 1 (January 2011), 1-11. DOI=10.1016/j.jnca.2010.07.006 http://dx.doi.org/10.1016/j.jnca.2010.07.006

KEY TERMS AND DEFINITIONS

Availability of Cloud Services: Model of cloud computing where resource availability is considered to be a strong constraint.

Cloud Computing Security: Model of cloud computing where security policies are designed and implemented to enforce protection of cloud resources both in soft and hardware form.

Cloud Computing: Describes a variety of computing concepts that involve a large number of computers connected through a real-time communication network such as the Internet. (Abu-Libdeh, Princehouse, & Weatherspoon, 2010). It is very similar to the concept of utility computing. In science, cloud computing is a synonym for distributed computing over a network, and means the ability to run a program or application on many connected computers at the same time.

Cloud Storage: Is a model of networked enterprise storage where data is stored in virtualized pools of storage which are generally hosted by third parties. Hosting companies operate large data centers, and people who require their data to be hosted buy or lease storage capacity from them.

Infrastructure as a Service: Offers cloud computing resources as an infrastructural service to external services requiring infrasture to run.

Reliability of Cloud Services: Fault tolerant model of cloud computing where resource reliability in terms of service delivery is considered to be a strong constraint.

Storage as a Service: Model of cloud computing where the storage resource availability is considered to be a strong constraint.

ENDNOTES

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