TAMING THE UNCERTAINTY OF PUBLIC CLOUDS
Maxim Schnjakin, Christoph Meinel
Hasso Plattner Institute
Potsdam University, Germany
Prof. Dr-Helmert-Str. 2-3, 14482 Potsdam, Germany
maxim.schnjakin, office-meinel@hpi.uni-potsdam.de

Abstract
Public cloud storage services enable organizations to manage data with low operational expenses. However, the benefits come along with challenges and open issues such as security, reliability, performance unpredictability and the risk to become dependent on a provider for its service. In our work, we presented a system that improves availability, confidentiality and reliability 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.

Recently, we conducted a proof-of-concept experiment for our application to evaluate the performance and cost effectiveness of our approach. We deployed our application using eight commercial cloud storage repositories in different countries. We observed that our implementation improved the perceived availability and, in most cases, the overall performance when compared with cloud providers individually. We also observed a general trend that cloud storage providers have constant throughput values - whereby the individual throughput performance differs strongly from one provider to another. With this, the experienced transmissions can be utilized to increase the throughput performance of the upcoming data transfers. The aim is to distribute the data across providers according to their capabilities utilizing the maximum of the available throughput capacity. To assess the feasibility of the approach we have to understand how providers handle high simultaneous data transfers. Thus, we put an additional focus on the performance and the scalability evaluation of those cloud storage providers, which are supported by our application.

1. 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 [12], [6].

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 a recent study by IDC the IT spending for public and private cloud storage will be over $20 billion by 2015 [16]. 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. Another review [15] states that, the amount of digital information created and replicated will grow by a factor of 300, from 130 exabytes to 40,000 exabytes, or 40 trillion gigabytes from 2005 to 2020. In addition, the authors estimate that the amount of digital information will double every two years.

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 [17]. 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 cloud service 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 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 [10], [11]) and implies significant higher storage and bandwidth costs without taking into account the security concerns regarding the retention of data.

A more economic 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

doi: 10.29268/stcc.2013.1.2.3
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 Redundant Array of Independent Disks (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 (bankruptcy), 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 [29]. 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 [24]. 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 this paper we present 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. Further, based on the indepth evaluation of the performance and resilience qualities of individual clouds and the results obtained, we propose possible strategies to improve the overall performance of the Cloud-RAID system.

2. **CLOUD-RAID 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. 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 harddrives, 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 [30], [14]. 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).

- **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. 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 [26] 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 (e.g. European data protection law).

B. STORAGE REPOSITORIES

1) CLOUD STORAGE PROVIDERS: Cloud storage providers are modeled as a storage entity that supports six basic operations, shown in table I. 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 [18]. 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, Microsoft Azure Storage, HP Cloud Object Storage, Google Cloud Storage (EU and US) and Nirvanix SND. Further providers can be easily added.

2) SERVICE REPOSITORY: Until now, 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 [5]. 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 [26]). 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create(ContainerName)</td>
<td>creates a container for a new user</td>
</tr>
<tr>
<td>write(ContainerName, ObjectName)</td>
<td>writes a data object to a user container</td>
</tr>
<tr>
<td>read(ContainerName, ObjectName)</td>
<td>reads the specified data object</td>
</tr>
<tr>
<td>list(ContainerName)</td>
<td>list all data objects of the container</td>
</tr>
<tr>
<td>delete(ContainerName, ObjectName)</td>
<td>removes the data object from the container</td>
</tr>
<tr>
<td>getDigest(ContainerName, ObjectName)</td>
<td>returns the hash value of the specified data object</td>
</tr>
</tbody>
</table>

TABLE I STORAGE CONNECTOR FUNCTIONS
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 the historical performance plus the ratings in the service registries and is saved in a Reputation Object (introduced in our previous work [3], [2], [4]). By reading this object, we know a provider’s reputation concerning each performance parameter (e.g. has high response time, low price). Note, the performance parameter is the median value of the last 10,000 performed requests as the performance of provider’s APIs may change over time. The values are only valid for our location of infrastructure. To observe all changes based on a) time of usage, b) possible infrastructure updates performed by cloud service providers, and c) the location of infrastructure, every user is expected to perform a set of experiments to determine those factors. These experiments would consume some time and are subject of future work and development of the application.

With the available 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.

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 (see figure 1).

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 [20]. 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 chunks and from them recalculates the original data. Note, any subset of k chunks is sufficient to reconstruct the original object of size s [23]. The total size of all chunks (after encoding) can be expressed with the following equation: 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. Competitive storage providers claim to have SLAs ranging from 99.9% to 99.999% 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. The automated determination of the appropriate m and k values remains a subject of future work.

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 unit 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 size². 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.

² The description of a code views each data package as having w bits worth of data.
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. In some cases a higher overhead is needed to ensure the basic file management functionality. As we mentioned above, services differ in their usage. For example, until recently Amazon didn’t offer rename support and the only way to rename an object was to let the service connector upload the object with a new name and delete the old one. Unfortunately there is still no possibility to rename a bucket (container). Therefore, it requires the according S3 service-connector to create a new bucket and copy the contents of old bucket to a new one. As discussed earlier in this chapter all accesses to the cloud storage providers can be executed in parallel (see algorithm 1). As erasure codes alone do not satisfy the confidentiality guarantee we enable our users to encrypt data prior to transmission\(^3\) (the functionality of the security component is described in our previous work [19]). 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).

Algorithm 1 The workflow of the data distribution process.

Require: coding parameters user.codingParams, list of prioritized storage providers user.providerList

1: f \(\leftarrow\) getDataObject()
2: p \(\leftarrow\) user.codingParams()
3: packages[] \(\leftarrow\) encode(f, p) // encodes \(n = k+m\) data packages from a file \(f\) according to the defined parameters and stores them into a file-array
4: sk \(\leftarrow\) generateSecretKey()
5: for all parallel data package : packages[] do
6: fenc[] \(\leftarrow\) encrypt(data package, sk)
7: end for parallel
8: pl \(\leftarrow\) user.providerList
9: for all parallel file : fenc[] do
10: storageConnector[pli].write(file)
11: if digest(file) = storageConnector[pli].getDigest(file) then
12: transmissionLog[pli, file] \(\leftarrow\) true
13: else
14: transmissionLog[pli, file] \(\leftarrow\) false
15: end if
16: end for parallel

4) REASSEMBLING THE DATA: When the service receives a read request, the service component fetches \(k\) from \(n\) chunks (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 chunks 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.

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. This primarily concerns the deployment of content. Some providers may offer discounts for large volumes and lower bandwidth rates for off-peak hours (as it is the case with unused computing capacity, for example Amazons Spot instances4). In our approach we plan ahead and take advantage of these possible discounts to optimize the overall costs of data hosting. It seems also reasonable to suppose that, user might be interested in shifting the up and download of content to his own off-peak ours (on weekend or at night). In this case, the management service would delegate selected workloads to a system component named task scheduler.

4. 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.

A. METHODOLOGY

The experiment was run in Hasso Plattner Institute (HPI), which is located close to Berlin, Germany, over a period of over 377 (24x7) hours, in the middle of February 2013. 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 II provides a summary of the conducted experiment. We used test files of different sizes from 100 kB up to 1 GB, 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 APIconnection 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.

1) MACHINES FOR EXPERIMENTATION: We employed three machines for experimentation. Neither is exceptionally highend, 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

2) **EXPERIMENT SETUP:** Figure 7 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 encodes the incoming object into \( n = k + m \) chunks (see III-C2). Again, the reconstruction of the original data requires any subset of \( k \) shares \([23]\). In the next step, each chunk will be sent to a different storage repository - whereby all requests will be executed in parallel.

The second machine (the lower part of the graph in figure 7) uploads the entire data object to multiple storage providers. If the receiving storage providers do not contain the entire data object any more, it will download the required files from other providers. This means that in case of data upload, the transfer is only possible if the data object is being successfully transferred to the transfer destination.

3) **ERASURE CONFIGURATION:** In our experiment we make use of the Cauchy-Reed-Solomon algorithm for two reasons. First, according to Plank et al. \([22]\) 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 \([21]\) 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 paper, 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. The expected availability for the selected configuration can be calculated using the following formula:

\[
\text{availability}(k,m)=\sum_{i=0}^{k} \binom{k+m}{i} A_i \times (1-A_i)^{k+m-i}
\]

With this, the configuration \([4,1]\) results in a monthly up-time percentage value of 99.999% or a tolerated failure of 25.868 seconds per month (under the assumption that the average monthly up-time percentage value \( Ap \) is 99.9%).

B. **SCHEMES AND METRICS**

The goal of our test is to evaluate the performance of our approach. Mainly we are interested in the effective 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 7 is captured with two log entries (START and END).

1) **ERASURE OVERHEAD:** Due to the nature of erasure codes, each file upload and download is associated with a certain overhead. On one hand this overhead is caused by the redundant \( m \) packages, which have to be stored, uploaded and sometimes downloaded (in the events of failure). As stated in III-C2, 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.

2) **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 chunks). In the event of a mismatch the system will delete the corrupted data and initiate a reupload 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.

C. **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.
1) 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 1/4 * 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 a storage overhead of only 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 3. 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 [19] we showed, that the average performance overhead caused by data encoding is less than 1% 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 [19] and [27].

Figure 3 The computational overhead caused by erasure with different configurations and file sizes. The overall overhead increases with growing file size regardless of the defined m and k parameters for the erasure configuration. In general, the encoding step requires not more than 0.5 % of the entire data transmission process.

2) 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 nine clouds, which are: Amazon US, Amazon EU, Azure, Box, HP, Google EU, Google US, Nirvanix and Rackspace. The particular combinations are represented in table III.

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 1/4th 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 4a and 4d). 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 17.3 minutes, whereas the download of the same file is performed in less than 21 seconds, see figures 5b and 5d). Then again, Google US service improves its performance clearly with the growing size of data objects (see figures 4a and 5a). 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 to read speeds for data objects up to 1 MB, (see figures 4a and 4b). 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 5b). Similar phenomena have been observed by read requests. Microsoft Azure belongs to the leading providers for reading 100 kB data objects (see figure 4a and 4d) and falls back by reading 100 MB files (see figure 5d).

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 chunks) 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 CRA 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 settings 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 fastest cloud providers (see figure 5c). 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 4a, 4b and 4c). 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 5c and 5d).

There is also an observed connection between the throughput rate and the size of data objects. Charts 4a to 4f 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 100 MB data objects Amazon EU falls back on the fourth place (see figures 5b and 5d). 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 units, which decrease with the growing number of \( k \) data packages (see chapter IV-B1).

<table>
<thead>
<tr>
<th>Cloud-RAID</th>
<th>Provider Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR-A</td>
<td>Amazon EU, Amazon US, Azure, Nirvanix, Rackspace</td>
</tr>
<tr>
<td>CR-B</td>
<td>Amazon EU, Amazon US, Azure, Google EU, Rackspace</td>
</tr>
<tr>
<td>CR-C</td>
<td>Amazon US, Azure, Google EU, Nirvanix, Rackspace</td>
</tr>
<tr>
<td>CR-D</td>
<td>Amazon EU, Amazon US, Azure, Google EU, Nirvanix</td>
</tr>
<tr>
<td>CR-E</td>
<td>Amazon EU, Azure, Google EU, Google US, Nirvanix</td>
</tr>
<tr>
<td>CR-F</td>
<td>Amazon EU, Google EU, Google US, Nirvanix, Rackspace</td>
</tr>
<tr>
<td>CR-G</td>
<td>Amazon EU, Amazon US, Azure, Google EU, Google US</td>
</tr>
<tr>
<td>CR-H</td>
<td>Amazon EU, Amazon US, Google EU, Google US, Nirvanix</td>
</tr>
<tr>
<td>CR-I</td>
<td>Amazon EU, Azure, Google EU, Google US, Rackspace</td>
</tr>
<tr>
<td>CR-K</td>
<td>Amazon EU, BoxNet, Google EU, Google US, Nirvanix</td>
</tr>
<tr>
<td>CR-L</td>
<td>Amazon EU, Amazon US, BoxNet, Google EU, Google US</td>
</tr>
<tr>
<td>CR-M</td>
<td>Amazon EU, Amazon US, Azure, BoxNet, Google EU</td>
</tr>
</tbody>
</table>

**TABLE III CLOUD-RAID WITH K=4 AND M=1**

**D. 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 IV and V 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 IV and V 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.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Filesize in kB</th>
<th>100</th>
<th>500</th>
<th>1024</th>
<th>10240</th>
<th>102400</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR-B</td>
<td>0.15</td>
<td>0.55</td>
<td>1.07</td>
<td>10.21</td>
<td>101.61</td>
<td></td>
</tr>
<tr>
<td>CR-G</td>
<td>0.16</td>
<td>0.52</td>
<td>0.99</td>
<td>9.28</td>
<td>92.25</td>
<td></td>
</tr>
<tr>
<td>CR-I</td>
<td>0.15</td>
<td>0.55</td>
<td>1.07</td>
<td>10.21</td>
<td>101.61</td>
<td></td>
</tr>
<tr>
<td>CR [6,1]</td>
<td>3.61</td>
<td>4.12</td>
<td>4.78</td>
<td>16.50</td>
<td>133.69</td>
<td></td>
</tr>
<tr>
<td>Azure</td>
<td>0.11</td>
<td>0.53</td>
<td>1.08</td>
<td>10.74</td>
<td>117.42</td>
<td></td>
</tr>
<tr>
<td>Amazon/Google</td>
<td>0.13</td>
<td>0.59</td>
<td>1.19</td>
<td>11.74</td>
<td>117.21</td>
<td></td>
</tr>
<tr>
<td>Rackspace</td>
<td>0.17</td>
<td>0.86</td>
<td>1.76</td>
<td>17.58</td>
<td>175.78</td>
<td></td>
</tr>
<tr>
<td>Nirvanix</td>
<td>4.14</td>
<td>4.72</td>
<td>5.46</td>
<td>18.65</td>
<td>150.48</td>
<td></td>
</tr>
</tbody>
</table>

TABLE IV COSTS IN $ FOR 10,000 READS.

5 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.
5. PERFORMANCE OPTIMIZATION

As stated in chapter IV, the involvement of providers with different throughput and response time capabilities can influence the overall performance of the Cloud-RAID application in a negative way. Once again, this can be attributed to the fact that in our approach the transmission of an individual data object depends on the capabilities (e.g. throughput or response time) of all the providers involved into the data distribution process. With Cloud-RAID, a data object is completely transferred, when the last data package is successfully transferred to its destination (see figure 6a).

One possible solution, to improve the overall transmission performance would be to stripe the original data (after the encoding step) into slices and to distribute the load across providers according to their capabilities. The aim would be to ensure that the transmission ends at around the same time. More specifically, the duration of the transmissions should take approximately the same time. Figure 6b illustrates the proposed approach and compares it with the current implementation. The optimization is only applicable under the assumption that the transmission performance can be increased through simultaneous transfers. Applying this method, we would be able to increase the overall transmission performance by utilizing the maximum of the available throughput capacity of participating providers.

The traffic of the network determines the speed of data movements, thus we can’t predict the speed at the time of uploading and downloading data precisely. However, in our tests, we noticed that most providers have more or less constant throughput values. With this, the experienced transmissions can be utilized to estimate the size of individual data packages for the upcoming data transfers.

As mentioned above our system executes all API requests in parallel. Therefore, to be able to make assumptions about the feasibility of the approach we have to clarify three questions: First, can the transmission performance be increased through simultaneous writing accesses to particular cloud providers? Second, does the performance of providers remain constant while data transmission process (or does it degrade over time)? And third, how quickly can we interact with the APIs of the cloud storage services. To answer these questions we conduct a new set of experiments focusing on the performance abilities of individual clouds in terms of response time and resilience. The latter will help us to understand how individual providers handle high object counts of different sizes.

A. EXPERIMENT SETUP

To measure the performance and resilience of selected clouds we conducted two further experiments (by utilizing the same infrastructure as in our prior experiments presented in chapter IV):

- The first part of the experiment clarifies the question, if the transmission performance can be increased by dividing the data objects into chunks and their simultaneous upload to cloud providers. To make assumptions about the performance and resilience of the involvement of providers with different throughput and response time capabilities can influence the overall performance of the Cloud-RAID application in a negative way. Once again, this can be attributed to the fact, that in our approach the transmission of an individual data object depends on the capabilities (e.g. throughput or response time) of all the providers involved into the data distribution process. With Cloud-RAID, a data object is completely transferred, when the last data package is successfully transferred to its destination (see figure 6a).

One possible solution, to improve the overall transmission performance would be to stripe the original data (after the encoding step) into slices and to distribute the load across providers according to their capabilities. The aim would be to ensure that the transmission ends at around the same time. More specifically, the duration of the transmissions should take approximately the same time. Figure 6b illustrates the proposed approach and compares it with the current implementation. The optimization is only applicable under the assumption that the transmission performance can be increased through simultaneous transfers. Applying this method, we would be able to increase the overall transmission performance by utilizing the maximum of the available throughput capacity of participating providers.

The traffic of the network determines the speed of data movements, thus we can’t predict the speed at the time of uploading and downloading data precisely. However, in our tests, we noticed that most providers have more or less constant throughput values. With this, the experienced transmissions can be utilized to estimate the size of individual data packages for the upcoming data transfers.

As mentioned above our system executes all API requests in parallel. Therefore, to be able to make assumptions about the feasibility of the approach we have to clarify three questions: First, can the transmission performance be increased through simultaneous writing accesses to particular cloud providers? Second, does the performance of providers remain constant while data transmission process (or does it degrade over time)? And third, how quickly can we interact with the APIs of the cloud storage services. To answer these questions we conduct a new set of experiments focusing on the performance abilities of individual clouds in terms of response time and resilience. The latter will help us to understand how individual providers handle high object counts of different sizes.

A. EXPERIMENT SETUP

To measure the performance and resilience of selected clouds we conducted two further experiments (by utilizing the same infrastructure as in our prior experiments presented in chapter IV):

- The first part of the experiment clarifies the question, if the transmission performance can be increased by dividing the data objects into chunks and their simultaneous upload to cloud providers. To make assumptions about the performance and resilience of the involvement of providers with different throughput and response time capabilities can influence the overall performance of the Cloud-RAID application in a negative way. Once again, this can be attributed to the fact, that in our approach the transmission of an individual data object depends on the capabilities (e.g. throughput or response time) of all the providers involved into the data distribution process. With Cloud-RAID, a data object is completely transferred, when the last data package is successfully transferred to its destination (see figure 6a).

One possible solution, to improve the overall transmission performance would be to stripe the original data (after the encoding step) into slices and to distribute the load across providers according to their capabilities. The aim would be to ensure that the transmission ends at around the same time. More specifically, the duration of the transmissions should take approximately the same time. Figure 6b illustrates the proposed approach and compares it with the current implementation. The optimization is only applicable under the assumption that the transmission performance can be increased through simultaneous transfers. Applying this method, we would be able to increase the overall transmission performance by utilizing the maximum of the available throughput capacity of participating providers.

The traffic of the network determines the speed of data movements, thus we can’t predict the speed at the time of uploading and downloading data precisely. However, in our tests, we noticed that most providers have more or less constant throughput values. With this, the experienced transmissions can be utilized to estimate the size of individual data packages for the upcoming data transfers.

As mentioned above our system executes all API requests in parallel. Therefore, to be able to make assumptions about the feasibility of the approach we have to clarify three questions: First, can the transmission performance be increased through simultaneous writing accesses to particular cloud providers? Second, does the performance of providers remain constant while data transmission process (or does it degrade over time)? And third, how quickly can we interact with the APIs of the cloud storage services. To answer these questions we conduct a new set of experiments focusing on the performance abilities of individual clouds in terms of response time and resilience. The latter will help us to understand how individual providers handle high object counts of different sizes.

A. EXPERIMENT SETUP

To measure the performance and resilience of selected clouds we conducted two further experiments (by utilizing the same infrastructure as in our prior experiments presented in chapter IV):

- The first part of the experiment clarifies the question, if the transmission performance can be increased by dividing the data objects into chunks and their simultaneous upload to cloud providers. To make assumptions about the
B. SCHEMES AND METRICS

As mentioned above, we intend to evaluate the performance of cloud storage providers, which are currently supported by our application. More specifically, we want to observe the behavior of cloud providers when it comes to parallel transmission of high data counts. In this context we are also interested in response times and resilience properties of the APIs. Therefore, as in the first experiment we capture each state of the workflow depicted in figure 7 with two log entries (START and END).

1) Response Time: In general, the measure of the response time (latency) depends on the network proximity, reliability of the services, we let our test clients read infrequently 10% of the transferred data back from the provider (not the recent ones) and compare the hash values against the expected one. The result of each run is the time elapsed between the execution of the first write request until the last write request.

• With the second test we aimed to find out, whether simultaneous uploads influence each other and to what extent. In each sequence our test client generates n data objects of a fixed size and then transfers them simultaneously to a cloud provider. In this part of the experiment, we are interested in the average duration of the data transfer operations (reads and writes). Then again, to check the integrity of the transferred data, the test client reads 10% of randomly selected data back from the cloud and compares it against an expected value.

Figure 7 presents the workflow of the second experiment. All transferred data objects will be only deleted after the completion of the experiment, as the aim is to observe the performance of providers while filling up the repositories.

![FIGURE 7 PROCESS WORKFLOW OF THE "STRESS TEST" EXPERIMENT (BPMN DIAGRAM).](image)

2)Availability: Usually, the availability is defined as the constancy of performance during a series of simultaneous transfers; and the reliability of the provider over a long period of sustained operation rates.

For our test we executed a series of simultaneous write requests with data objects of various sizes (1 MB, 10 MB, 100 MB and 1 GB). We made the decision to start with the 1 MB file size due to our observations from the previous experiment IV. We observed that with Cloud-RAID, the transmission of smaller data objects (e.g. 64 kB, 100 kB, 500 kB) takes almost the time same as the transmission of a 1 MB file. The underlying reason is that execution of both read and write requests is dominated by erasure overhead, DNS lookup and API connection establishment time.

3) Availability: Applying to cloud storage services we define the perceived availability of providers as $\text{Availability} = \frac{\text{Up}}{\text{Total}}$. This definition of availability can be found in the SLAs of most storage providers. Indeed, some vendors use self-defined metrics for calculation of the availability of their services. Rackspace, for example, perceives its network to be down if user requests fail during two or more consecutive 90 second intervals$^6$. At the same time Google defines downtime when

---

$^6$ http://www.rackspace.com/cloud/legal/sla/
more than 5% of request failures occur in a certain time interval\(^7\). The latter availability metrics allow a higher margin for failures.

C. EMPIRICAL RESULTS

This section presents the results in terms of response time and resilience based on over one million requests. Due to space constraints, we present only some selected results from the conducted experiment.

1) **Response Time:** In order to observe the behavior of the participating storage providers we uploaded data units of various sizes to each provider (100 kB, 1 MB, 10 MB and 100 MB). Each transferred object has a unique hash value, regardless of file size. After that, we performed a series of randomized download requests and measured the time span between executed calls and received responses. In addition, we measured the time our system needed to calculate the hash value of data packages (locally).

<table>
<thead>
<tr>
<th>Provider</th>
<th>File Size (in kB)</th>
<th>API call (in msec)</th>
<th>Local hash calculation (in msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon [EU]</td>
<td>100</td>
<td>485</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>417</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>10240</td>
<td>463</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>102400</td>
<td>512</td>
<td>547</td>
</tr>
<tr>
<td>Amazon [US]</td>
<td>100</td>
<td>1326</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>1069</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10240</td>
<td>1280</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>102400</td>
<td>1390</td>
<td>550</td>
</tr>
<tr>
<td>Azure</td>
<td>100</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>10240</td>
<td>26</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>102400</td>
<td>26</td>
<td>547</td>
</tr>
<tr>
<td>Box</td>
<td>100</td>
<td>308</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>296</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>10240</td>
<td>291</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>102400</td>
<td>317</td>
<td>1258</td>
</tr>
<tr>
<td>Google</td>
<td>100</td>
<td>85</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>71</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10240</td>
<td>60</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>102400</td>
<td>63</td>
<td>548</td>
</tr>
<tr>
<td>Nirvanix</td>
<td>100</td>
<td>380</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>393</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>10240</td>
<td>390</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>102400</td>
<td>408</td>
<td>547</td>
</tr>
<tr>
<td>Rackspace</td>
<td>100</td>
<td>171</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>152</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10240</td>
<td>169</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>102400</td>
<td>194</td>
<td>548</td>
</tr>
</tbody>
</table>

**TABLE VI THE COMPARISON OF HASH VALUE CALCULATIONS**

The results of the experiment are presented in table VI. There are few observations that can be taken from the table:

1) Box uses a different hash method, therefore it takes our system nearly twice as much time to calculate the hash values;
2) The measured time span is not affected by the size of a data unit;
3) API calls for receiving hash values of larger data units (greater-than 100 MB) is faster than their on site calculation;
4) From 2 and 3 we conclude that each provider stores the information to a meta-object after computing the hash value of the received data unit.

Following this we conclude that on any API getHash-call the requested information is extracted from the meta-object and transferred to the caller.

Further we observed a general trend that our test clients experienced consistent and constant response times in most cases - whereby the individual latency values differ extremely from provider to provider. After the first analysis we divided providers into three clusters related to our test location:

- Fast (response time <200ms);
- Medium (response time varies between 200 and 1500ms); and
- Slow (response time >1500ms)

The figures 8 and 9 show how quickly providers react on a getHash request. At several time instances during the experiment we observed increased response time which can be attributed to the sudden increases in request traffic on the target server nodes (for example in case of Google-US service in figure 8a). Overall, the best and continuous provider for our location of infrastructure is Azure. The average time needed by the service to react on a request is about 50 milliseconds (see figure 8a). We measured the slowest reaction time on Amazon-US service (see figure 9). The reason is obvious and refers to the large distance between the location of our test clients (located in Germany) and the destination server. With a deviation of up to 100
milliseconds the providers Google and Amazon do not provide as constant results than other providers. It would be speculative to explain the experienced behavior. One reason could be, that both Amazon and Google are running more cloud services at the same nodes than other providers, which would result in an additional traffic load on servers. Then again, the behavior might be also related to the usage of different consistency models, which is the subject of our further analysis.

However, the empirical results summarized in this section are based on continuous monitoring over the course of over 20 days. Overall, the measured values appear to be sufficiently comprehensive in order to effectively predict the response time for upcoming requests. Hence, the information can be used for an intelligent data placement within Cloud-RAID application.

2) Resilience: For the sake of brevity, we present only some selected results from the conducted experiment. In the first instance, the performance comparison focusses on the upload performance of six clouds presented earlier (Amazon, Box, Google, Nirvanix, HP and Rackspace). The evaluation of download performance showed similar results.

The first part of the experiment can be briefly summarized as follows: a data unit F of size s(F) was splitted into multiple chunks of equal size (starting with 5 and ending with 100 in intervals of 5 segments) and transferred to a cloud. At the same time, we measured the time span between the first and the last read or write request within each segmentation interval. Here again, our system tries to execute all API requests in parallel.

Before the experimentation, we assumed that the transmission performance increases with simultaneous
chunk transfer. Hence, the interpretation of the results of the conducted experiment focuses on the following values:

- the number of chunks where the increase in performance stops, we will denote the value as (maximum) segmentation level \( x \);
- the size of chunks at the segmentation level with the best performance value, and which we denote as \( s(Fx) = \frac{s(F)}{x} \);
- the average transmission performance of all chunks at the segmentation level \( x \), which we denote as \( P(Fx) \);
- the transmission performance of chunks of size \( s(Fx) \) in case of a native data transfer, denoted as \( P(s(Fx)) \). This means that an individual chunk is transferred as a single file with one single thread (the segmentation level equals to one);
- the relation between the transmission performance of a native data transfer\(7\) \( P(F) \) and \( P(Fx) \), expressed as an improvement factor;
- the relation between the transmission performance of a native transfer of a single chunk \( P(s(Fx)) \) and \( P(Fx) \);

Table VII captures the results of the experiment. The evaluation confirmed the assumption, that the behavior of providers differs when it comes to simultaneous transfers of a large number of data objects. In general, an increase in segmentation level goes hand in hand with an increase in the transmission performance, at least to a certain extent. As we have discussed earlier, vendors have optimized their infrastructure for particular file sizes. Hence, the improvement factor depends of the file size and varies from one provider to another. More specifically, relatively slow providers with optimized APIs for transmissions of smaller data objects, achieve significantly better performance with a higher segmentation rate, as the size of individual chunks decreases with an increasing number of segments. For example, a native transmission of a 100 MB data object to Nirvanix takes about 82.13 seconds (see 12)e. The transmission of the same object in 75 segments (with a size of 1.33 MB each) takes nearly 8.9 seconds, which improves the transmission rate by a factor of 9.2. In terms of performance of writing a 10 MB file, Nirvanix achieves only an improvement factor of 5.46 (see table VII). HP Cloud Storage service shows similar behavior. A native transfer of a 100 MB file takes the service about 45 minutes, whereas the uploaded content takes only 41 seconds.

Similar behavior can also be observed by providers with higher throughput rates, although less pronounced. At one extreme, Amazon achieves the highest improvement factor of 32, when it comes to an upload of a 100 MB file in 85 segments. However, Google-EU provides consistent high data throughput for all data sizes and therefore achieves only an improvement factor of 5 for a segmented transmission of 100 MB data objects.

Important to note, is also the relation between the performance of a native chunk transfer \( P(s(Fx)) \) and the performance of multiple simultaneous chunk transfers (cumulated transfer) of the same size \( P(Fx) \). For fast providers (e.g. Amazon and Google), the values of \( P(s(Fx)) \) are approximately identical to \( P(Fx) \) (see table VII). This means, the choice of chunk size determines the accumulated transmission performance of the original data. In order to minimize transmission times of unsegmented data objects we have to identify the optimal chunk size. Experienced deviations can be attributed to the overhead associated with an establishment of API connections. In addition, with a high number of threads, it cannot be guaranteed, that all processes are executed exactly in parallel. Further, no assumptions can be made about the order in which individual API connections are processed on the side of providers. It is important to note, that the transmission of chunks of small sizes takes only few seconds, so that minor delays in the thread processing affect the measurement results.

For other providers the relation between \( P(s(Fx)) \) and \( P(Fx) \) may differ up to 300% (see table VII). The observed behavior could be attributed to weaker load balancing capabilities. It could also be assumed, that these providers limit the throughput performance beyond a certain number of connections that are opened simultaneously. Here again, the minimization of native transfer time requires the identification of an appropriate chunk size and in this case the upper limit of simultaneous connections.

The evaluation of the second test provided insights into the constancy of performance during simultaneous data transfers. The results of the experiment are presented in figure 10. Following a preliminary analysis, the general behavior of cloud storage providers can be grouped into three categories:

- the number of simultaneous transfers has no impact on the average transmission performance of individual chunks (see figure 10a, 10b, 10c);
- additional connections decrease the average performance (e.g. in case of Google or Nirvanix in figure 10e and 10f)
- the average performance decreases above a certain segmentation level, the behavior is shown in figure 10d

An especially interesting point is the constancy of performance during simultaneous transfers observed at various segmentation levels during the experiment. Figure 11 shows that the transmission performance is relatively constant in most cases, except the providers Box, Nirvanix and HP. The average throughput performance of the latter services decreases above a certain number of simultaneous transfers. For example, at the segmentation level of 90 the transmission of threads 51 to 90 takes twice as much time as the transmission of threads 1 to 50 (see figure 11d). Again, the behavior might be related to a server-sided limitation of throughput performance beyond a certain number of open connections and is subject of the further analysis. However,
the decreasing performance of threads as the number of threads increases, which is observed in individual cases, can be compared with processor contention on virtual machines. As the load raises above a certain limit, the amount of bandwidth that each thread receives decreases. But as the figure 12 shows, the higher number of threads leads to a significantly higher bandwidth than individual threads.

From these observations, we come to the following decisive conclusions. Simultaneous transfers can increase significantly the transmission performance of individual data objects. The improvement factor depends on the capabilities of cloud providers when dealing with different file sizes and simultaneous transfers. A "one-size-fits-all" approach is not workable. The optimization of Cloud-RAID requires an intelligent identification of chunk size taking into account server-sided limitations of open API connections. Nevertheless, conducted experiments provide sufficient results for the identification of an appropriate transfer strategy based on the individual capabilities of cloud storage providers.

3) Availability: During the second part of the experiment (resilience testing) we performed over 660,000 operations (read/write). In this time period we observed only a few number of failed requests. After a thorough evaluation of the occurred failures, we can safely say, that nearly all exceptions can be attributed to the implementation errors on our side. Nevertheless, we experienced a number of operations that cloud not be completed due to some error on the server side (e.g. readTimeOut or peerNotAuthenticated). Table VIII presents the observed availability of all experiments calculated as $\frac{\text{successful reads}}{\text{total reads}}$. Further, the table captures also the results of the infrequent hash value comparison, which was successful in nearly all cases, except the providers Rackspace and Box. Note, the observed
availability values represent only a short period of time, which is little more than twenty days. Actual values may differ from our observations.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Number of writes</th>
<th>Errors</th>
<th>(\frac{\text{writes completed}}{\text{writes tried}})</th>
<th>wrong hash value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>72500</td>
<td>0</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>Amazon</td>
<td>72500</td>
<td>16</td>
<td>99.978%</td>
<td>0</td>
</tr>
<tr>
<td>Nirvanix</td>
<td>42000</td>
<td>3</td>
<td>99.993%</td>
<td>0</td>
</tr>
<tr>
<td>Rackspace</td>
<td>42000</td>
<td>145</td>
<td>99.643%</td>
<td>5</td>
</tr>
<tr>
<td>Box</td>
<td>42000</td>
<td>0</td>
<td>99.905%</td>
<td>40</td>
</tr>
<tr>
<td>HP</td>
<td>42000</td>
<td>0</td>
<td>100%</td>
<td>0</td>
</tr>
</tbody>
</table>

TABLE VIII THE OBSERVED AVAILABILITY DURING THE EXPERIMENT.

6. RELATED WORK

The main idea underlying our approach is to provide RAID technique at the cloud storage level. In [9] 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) [7] and proofs of data possession (PDP) [7] and unifies these two approaches. In comparison to our work, HAIL requires storage providers to run some code whereas our system deals with cloud storage repositories as they are. Further, HAIL does not provide confidentiality guarantees for stored data. In [13] 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 [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 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 [11] 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 [8] protocol improves availability and integrity of cloud-stored data by replicating 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 guarantees, 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.

7. 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 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). Our approach makes use of erasure code techniques for stripping 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. More specifically, 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. We also observed that slow vendors may influence the transmission performance in a negative way. On the lookout for possible performance optimizations of Cloud-RAID we conducted further experiments which helped us to understand how individual clouds handle high object counts of different sizes. The tests focused on the performance evaluation in terms of service provider’s response time and its resilience (i.e. availability, performance).

The results clearly demonstrate that simultaneous transfers can increase significantly the transmission performance (depending on the individual capabilities of cloud providers when dealing with different file sizes and simultaneous transfers). The requirements of implementing the proposed optimization include the identification of: appropriate chunk size (based on the individual throughput capabilities of each provider), and a limitation of open API connections (i.e. load balancing).

Nevertheless, we do not find one winning strategy to optimize the performance of Cloud-RAID. Rather, the optimization needs to be tackled individually per provider when it comes to simultaneous transfers of high object counts. Performance tests showed that our system is best utilized for deployment of large files. In case of transmission of smaller data objects the transmission is highly affected by the overhead which is associated with DNS look-ups, API connection time, and API handling of multiple threads. With increasing segmentation level (smaller chunk size), response time becomes significant as it can dominate the overall transmission rate. Therefore, it is an important factor when deciding on a segmentation strategy.

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 certainly 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. Google’s Durable Reduced Availability (DRA) storage could be a pioneer for such services, as it provides users storage buckets at lower cost (up to 29% less than standard Google Cloud Storage) and lower availability (99% instead of 99.9%).

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.

8. 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 behaviour might be related to the usage of different consistency models and is subject of further analysis.

The performance of the model introduced heavily depends on relative location of cloud service providers and the uploading unit - Cloud-RAID. Therefore, the actual use is based on the assumption that the location of clouds and the uploading unit is always the same. In the course of future work, we aim to address an alternative scenario, where the parties involved into the distribution process do not need to be at the same location (e.g. upload can be located in Europe for saving data where as the downloading user can be in the US).

In addition, we are also planning 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.

9. References

[27] David Sarno. Microsoft says lost sidekick data will be restored to users. Los Angeles Times, October 2009.

Authors

Maxim Schnjakin studied computer science at the University of Trier and moved to Hasso-Plattner-Institute after graduation. Since 2008 he works as a scientific assistant at the chair of professor Ch. Meinel. The main focus of his research is enforcement of high availability and security requirements in public clouds.

Christoph Meinel studied Mathematics and Computer Sciences at the Humboldt-University of Berlin from 1974 to 1979 and received his PhD degree (Dr. rer. nat.) in 1981.
Today, Christoph Meinel is President, Scientific Director and CEO of the Hasso-Plattner-Institut for IT-Systems Engineering (HPI). Internet-Technologies and Systems are in the center of research and teaching of his chair.