Scrutinizing the State of Cloud Storage with Cloud-RAID: A Secure and Reliable Storage Above the 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 and the risk to become dependent on a provider for its service. In our previous 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 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, in this paper we focus on the performance and the scalability evaluation of particular cloud storage providers. To this end, we deployed our application using eight commercial cloud storage repositories in different countries and conducted a set of extensive experiments.

I. Introduction

The usage of 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 researchers argue for the benefits of cloud computing focusing on the economic value [8], [2].

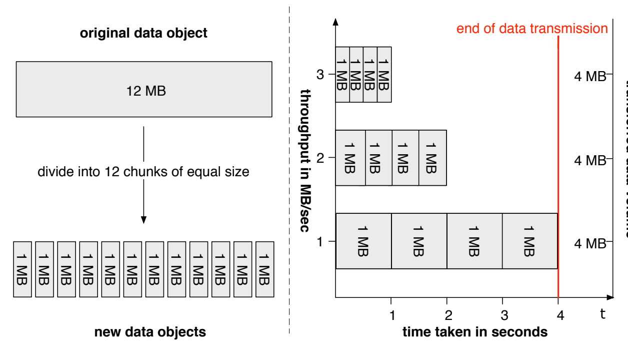
However, despite of the non-contentious financial advantages cloud computing raises questions about privacy, security and reliability. Among available cloud offerings, storage services reveal an increasing level of market competition. According to iSuppli [6] global cloud storage revenue is set to rise 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.

For a customer (service) to depend solely on one cloud storage provider has its limitations and risks. In general, vendors do not provide far reaching security guarantees regarding the data retention [11]. Placement of data in the cloud removes the physical control that a data owner has over data. So there is a risk that a service provider might share corporate data with a marketing company or use the data in a way the client never intended.

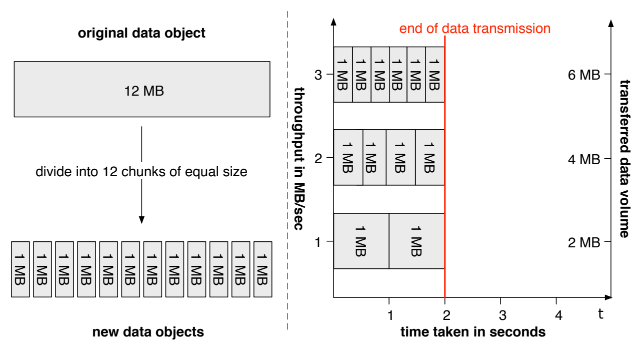
Further, customers of a particular provider might experience vendor lock-in. In the context of cloud computing, it is a risk for a customer to become dependent on a provider for its services.

In our previous work [12], [18] and [17] we presented an approach that deals with the mentioned problems by separating data into unrecognizable slices, that are distributed to different providers. It is important to note, that only a subset of the providers needs to be available in order to reconstruct the original data. This is indeed very similar to what has been done for years at the level of file systems and disks.

In recent months we conducted extensive experiments for our application to evaluate the overall performance and cost effectiveness of the approach. The results are presented in our last work [16]. We observed that, with an appropriate coding configuration, our system is able to improve significantly the performance of the data transmission process. Nevertheless, we also observed that storage providers differ extremely in their upload and download capabilities. In addition, some vendors seem to have optimized their infrastructure for large files, while others focused more on smaller data objects. However, the involvement of providers with different throughput and response time capabilities might influence the overall performance of our application in a negative way. This is due 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 (Redundant Array of Inexpensive Disks), a data object is completely transferred, when the last data package is successfully transferred to its destination (see figure 1a). On the lookout for solutions, to improve the overall transmission performance we decided on the following



(a) The current implementation of Cloud-RAID. Slow providers hamper the overall transmission performance.



(b) The optimization step utilizes the maximum of provider's throughput capacity, in such a way that individual chunk transfers finish approximately at the same time.

Figure 1. Optimization of Cloud-RAID.

approach. The idea is to stripe the original data (prior to the encoding step) into slices - *chunks* - and to distribute the load across providers according to their capabilities. The aim would be to ensure that the transmission ends at the same time. More specifically, the duration of the transmissions should take approximately the same time. Figure 1b illustrates the described approach. The drawback of the approach is that providers with higher throughput rates would receive more and more data over time. However, 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.

In our tests, we noticed that most providers have constant throughput values. With this, the experienced transmissions can be utilized to estimate the size of individual data packages for the upcoming data transfers.

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 experiment focusing on the performance abilities of cloud storage providers in terms of response time and resilience. The latter will help us to understand how individual providers handle high object counts of different sizes.

The main contributions of this paper is an experimental study on a world-wide testbed of Cloud-RAID application focussing on performance and resilience evaluation of individual cloud storage providers.

II. The Cloud-RAID System

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 administration responsibilities from the customer's side. As mentioned before, 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. In order to achieve our goal we foster the usage of erasure coding techniques (interested readers will find more information in our previous work [18]).

Cloud-RAID is implemented using Grails, JNI and C technologies, with a MySQL back-end to store user accounts, current deployments, meta data, the capabilities and the pricing of cloud storage providers. Keeping the meta data locally ensures that no individual provider will have access to stored data. In this way, only users that have authorization to access the data will be granted access to the shares of (at least) k different clouds and will be able to reconstruct the data. Further, our implementation makes use of AES for symmetric encryption, SHA-1 and MD5 for cryptographic hashes and an improved version of Jerasure library [13] for using the Cauchy-Reed-Solomon and Liberation erasure codes. The usage of erasure coding techniques enables us to tolerate the loss of one or more storage providers without suffering any loss of content [10], [19]. 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. Further details can be found in our previous work [17], [15] and [14].

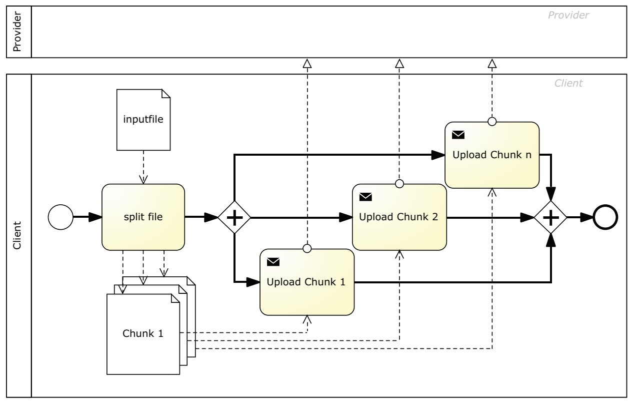


Figure 2. Process workflow of the experiment (BPMN diagram).

III. The State of Cloud Storage

In this section we present an evaluation of the conducted experiment that aims to clarify the main questions concerning the performance and resilience aspects of storage providers when our application is used to store data on public clouds.

A. Experiment Setup

The testing was run in Hasso Plattner Institute (HPI), which is located close to Berlin, Germany, over a period of over 480 hours, in November 2012. As it spans twenty 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 and thirteen physical locations in both Europe and USA. The experiment time comprises ten rounds, with each round consisting of a set of predefined test configurations (in the following sequences).

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 sent. 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. In this way each client waits for a think time before the next request is generated.

1) Machines for Experimentation: We employed two machines for experimentation. Neither is exceptionally high-end, but each represents a middle-range commodity processor, which 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.

B. Methodology

To measure the performance and resilience of selected clouds we conducted two series of experiments:

- 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 reliability of the services, we let our test client read infrequently 10% of the transferred data back from the provider and compare the hash values against an expected date. 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 2 presents the workflow of the 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.

C. 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, we implemented a simple logger application to record the results of our measurements. In total we log 21 different events. For example, each state of the workflow depicted in figure 2 is captured with two log entries (START and END).

1) Response Time: In general, the measure of the response time (latency) depends on the network proximity, congestions in network path and traffic load on the target server. We define response time as a time delay a storage provider needs to react on an API call. More precisely, we want to measure the time interval between starting the API call to download a file and the receipt of its first byte. To this end we contacted the respective providers with a request for instruction to perform the correct measurement. Thereupon we received nearly the same answers: we were recommended to send an API call and to track the time span until the network adapter will receive the first bits.

The implementation of the recommended procedure seemed to complicated. Therefore we looked for an easier and simpler approach to measure something that would reflect the workload and the response time of a provider. We concluded that we could use the $getHash$ method for two reasons: first, based on our observations (see below) the hash value is computed only ones after the data has been received by a storage provider. Secondly, the size of the data packet with the requested information is small enough to be ignored. Note, that the procedure does not exclude the amount of time that a storage provider spends processing the request. Nevertheless, as each provider is expected to process the requests in the same way, we can presume that the approach described above will reflect the response time of providers with sufficient precision.

2) Resilience: In general, resilience is defined as the ability of a system (network, service, infrastructure, etc) to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation. In the context of our experiment we assess the resilience of a provider based on the following:

1) The constancy of performance during a series of simultaneous transfers; and
2) 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 [16]. We found out, that with Cloud-RAID, the transmission of smaller data objects (e.g. 64 kB, 100 kB, 500 kB) takes almost the same time 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: Usually, the availability is defined as $\frac{uptime}{uptime+downtime}$. Applying to cloud storage services we define the perceived availability of providers as $\frac{number\_of\_successful\_requests}{number\_of\_all\_requests}$. 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[1]. At the same time Google defines downtime when more than 5% of request failures occur in a certain time interval[2]. The latter availability metrics allow a higher margin for failures.

D. Empirical Results

This section presents the results in terms of response time and resilience based on over 800.000 requests. Due to space constraints, we present only some selected results from the conducted experiment.

[1]http://www.rackspace.com/cloud/legal/sla/

[2]https://developers.google.com/storage/docs/sla

Provider	File Size (in kB)	API call (in msec)	Local hash calculation (in msec)
Amazon [EU]	100	485	0
	1024	417	4
	10240	463	54
	102400	521	547
Amazon [US]	100	1326	0
	1024	1069	5
	10240	1280	54
	102400	1390	550
Azure	100	28	0
	1024	36	4
	10240	26	54
	102400	26	547
Box	100	308	1
	1024	296	12
	10240	291	124
	102400	317	1258
Google	100	85	0
	1024	71	5
	10240	60	54
	102400	63	548
Nirvanix	100	380	0
	1024	393	4
	10240	390	54
	102400	408	547
Rackspace	100	171	0
	1024	152	5
	10240	169	54
	102400	194	548

Table I
THE COMPARISON OF HASH VALUE CALCULATIONS

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).

The results of the experiment are presented in table I. 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

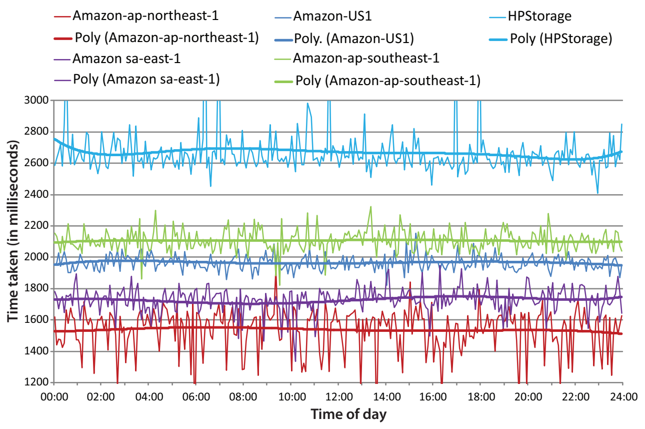


Figure 4. Average response time of the providers Amazon and HP over a period of twenty days.

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 3 an 4 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 3a). Overall, the best and continuous provider for our location is Azure. The average time needed by the service to react on a request is about 50 milliseconds (see figure 3a). We measured the slowest reaction time on Amazon-US service (see figure 4). The reason is obvious and refers to the largest 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 cloud 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 CloudRAID application.

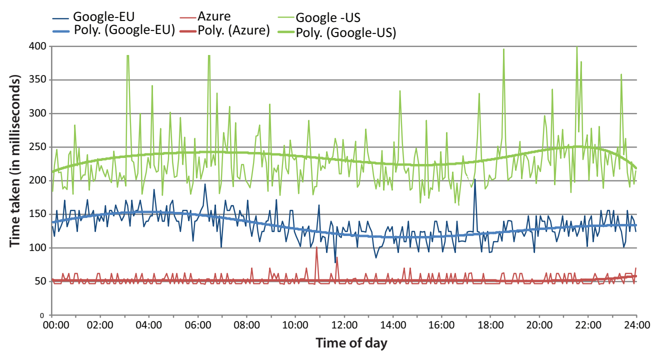
2) Resilence: Due to space constraints, we present only some selected results from the conducted experiment. In the first instance, the performance comparison focusses on the upload performance of six clouds, they are: Amazon EU, Box, Google EU, Nirvanix, HP Cloud Storage and Rackspace. The evaluation 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. In general, our system follows a model of one thread per chunk in such a way that all provider accesses were executed in parallel.

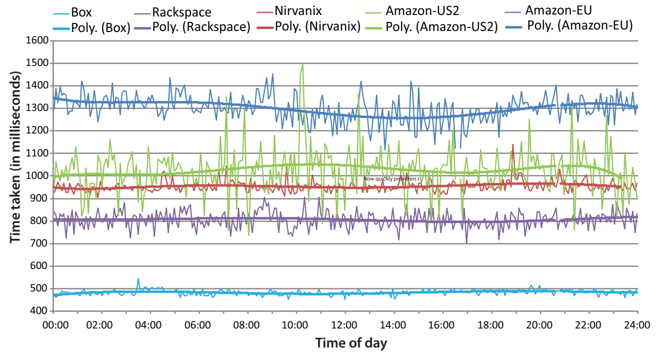
Before the experimentation, we assumed that the transmission performance increases with simultaneous chunk transfer. Hence, the interpretation of the results of the conducted experiment focusses on the following values:

- 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 $s(F_x)$ ($= \frac{s(F)}{x}$);
- the average transmission performance of all chunks at the segmentation level x, which we denote as $P(F_x)$;
- the transmission performance of chunks of size $s(F_x)$ in case of a $native$ data transfer, denoted as $P(s(F_x))$. This means that an individual file is transferred as a single file with one single thread;
- the relation between the transmission performance of a native data transfer $P(F)$ and $P(F_x)$, expressed as an $improvementfactor$;
- the relation between the transmission performance of a native transfer of a single chunk $P(s(F_x))$ and $P(F_x)$;

Table II 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



(a) Response time of the providers: Google and Azure



(b) Response time of the providers: Amazon, Box, Nirvanix and Rackspace

Figure 3. Average response times of cloud storage providers over a period of twenty days. The representation uses the trend line feature of excel to calculate a polynomial trend line (grade 6), which "straightens" the values. In general, we attach less importance to outliers in our experimentation.

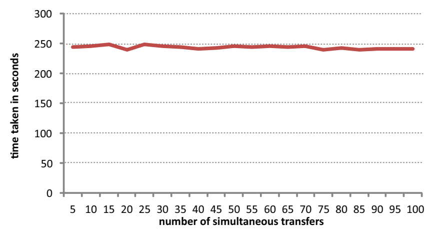
Provider	Native transfer		Best segmentation level x	Simultaneous transfer		Improvement factor	Single chunk transfer	
	$s(F)$ in MB	$P(F)$ in ms		$s(F_x)$ in MB	$P(F_x)$ in ms	$\frac{P(F)}{P(F_x)}$	$s(F_x)$ in MB	$P(s(F_x))$ in ms
Amazon	1	1941	50	0.02	493	3.94	0.02	422
	10	14242	85	0.12	920	15.48	0.12	731
	100	122922	85	1.18	3785	32,48	1,18	2257
	1000	1265086	100	10.00	18590	68.05	10.00	17872
Google	1	2514	85	0.01	2436	1.03	0.01	2208
	10	3812	25	0.40	2449	1.56	0.40	2393
	100	20049	35	2.86	3906	5.13	2.86	3322
	1000	154327	80	12.50	16318	9.46	12.50	12317
Nirvanix	1	4597	5	0.20	3049	1.51	0.20	636
	10	22276	10	1.00	4083	5.46	1.00	1443
	100	821391	75	1.33	8924	92.04	1.33	3409
Rackspace	1	11383	30	0.03	2230	5.10	0.03	745
	10	103634	55	0.18	7157	14.48	0.18	2400
	100	1038696	100	1.00	18072	57.48	1.00	11004
HP	1	25513	50!	0.02!	3099!	8.23!	0.02!	803!
	10	230918	95	0.11	8397	27.50	0.11	3047
	100	2683488	100	1.00	41047	65.38	1.00	23626
Box	1	18561	50	0.02	10986	1.69	0.02	1445
	10	145346	50	0.20	9458	15.37	0.20	3777
	100	1393105	50	2.00	36781	37.88	2.00	30383

Table II
THE TABLE CAPTURES THE RESULTS OF THE EXPERIMENT. DATA OBJECTS WERE SPLITTED INTO MULTIPLE CHUNKS OF EQUAL SIZE AND TRANSFERRED TO CLOUD STORAGE PROVIDERS.

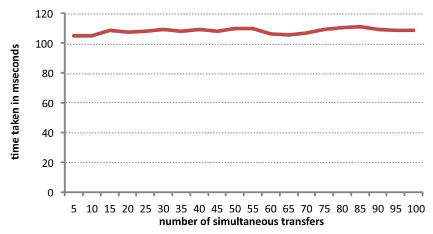
with an increasing number of segments. For example, a native transmission of a 100 MB data object to Nirvanix takes about 82,13 seconds. The transmission of the same object in 75 segments (with a size of 1,33 MB each) takes nearly 2,6 seconds, which improves the transmission rate by a factor of 92. In terms of performance of writing a 10 MB file, Nirvanix achieves only an improvement factor of 5,46 (see table II). HP Cloud Storage service shows similar behavior. A native transfer of a 100 MB file takes the service about 45 minutes, whereas the segmented upload takes only 23,6 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.

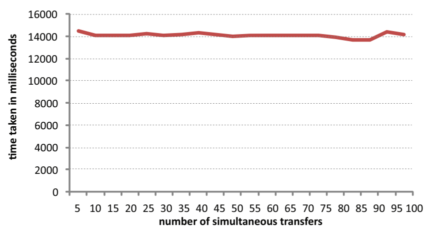
Important to note, is also the relation between the performance of a native chunk transfer $P(s(F_x))$ and the performance of multiple simultaneous chunk transfers (cumulated transfer) of the same size $P(F_x)$. For fast providers (e.g. Amazon and Google), the values of $P(s(F_x))$ are approximately identical to $P(F_x)$ (see table II). 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 establish-



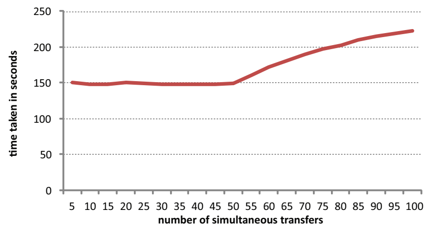
(a) HP



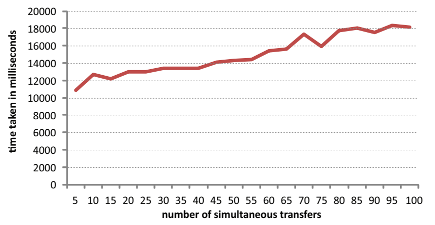
(b) Rackspace



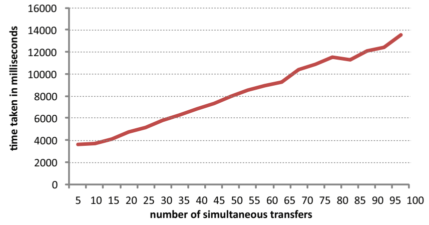
(c) Amazon



(d) Box

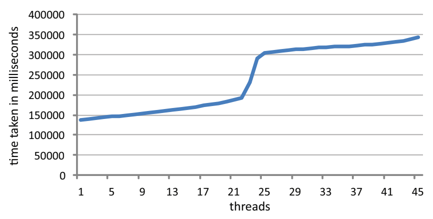


(e) Nirvanix

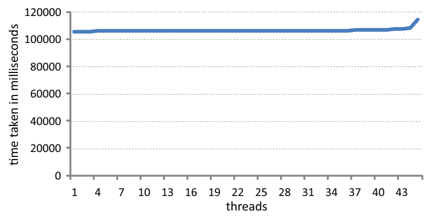


(f) Google

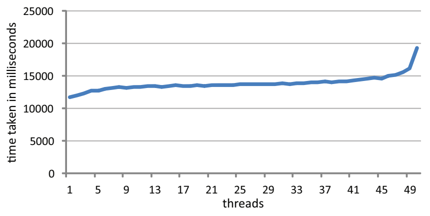
Figure 5. Average transmission performance of simultaneous uploads with increasing number of chunks. Each chunk has a size of 10 MB.



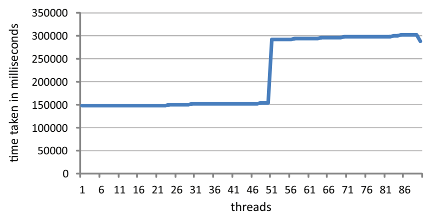
(a) HP.



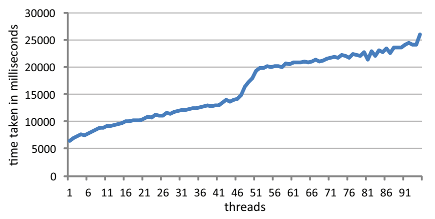
(b) Rackspace.



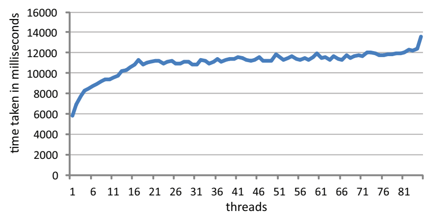
(c) Amazon.



(d) Box.



(e) Nirvanix.



(f) Google.

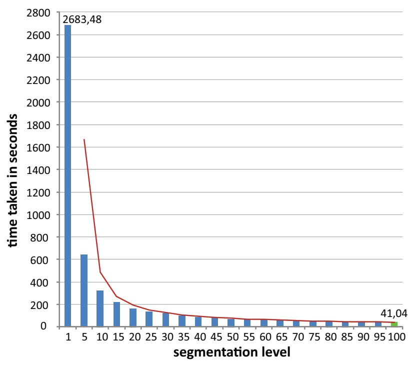
Figure 6. Transmission performance of single threads at different segmentation levels. Each thread transfers a chunk of 10 MB.

ment 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.

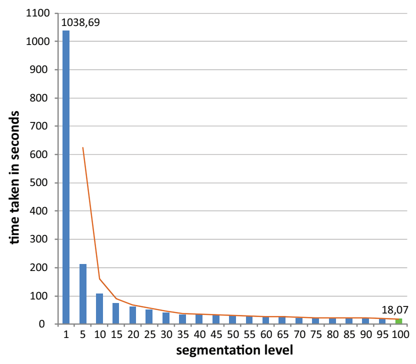
For other providers the relation between $P(s(F_x))$ and $P(F_x)$ may differ up to 300% (see table II). The observed behavior could be attributed to weaker load balancing capabilities. It cloud be also presumed, that these providers limit the throughput performance beyond a certain number of connections 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 5. Following a preliminary analysis, the general behavior of cloud storage providers can be grouped into three categories:

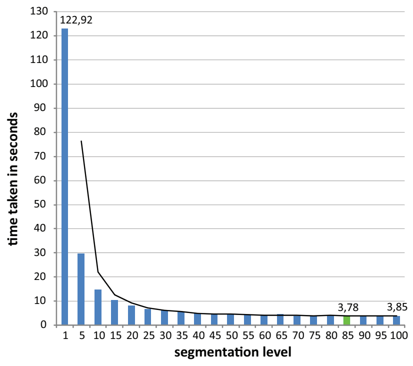
- the number of simultaneous transfers has no impact on the average transmission performance of individual chunks (see figure 5a, 5b, 5c);
- additional connections decrease the average performance (e.g. in case of Google or Nirvanix in figure



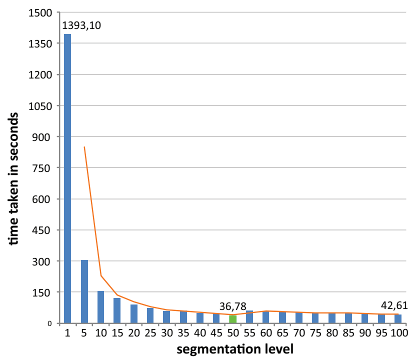
(a) HP



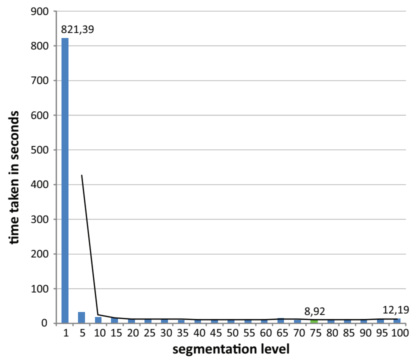
(b) Rackspace



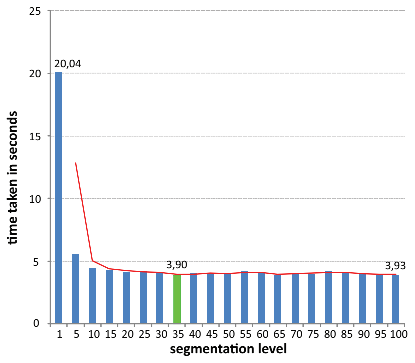
(c) Amazon



(d) Box



(e) Nirvanix



(f) Google

Figure 7. Average transmission performance in seconds observed on all writes at different segmentation levels. Each initial data object (an unsegmented file) has a size of 100MB. The highlighted bars correspond to the best segmentation levels, which represent the highest factor of improvement for individual providers. The size of chunks decreases with growing segmentation level.

5e and 5f)

- the average performance decreases above a certain segmentation level, the behavior is shown in figure 5d

An especially interesting point is the constancy of performance during simultaneous transfers observed at various segmentation levels during the experiment. Figure 6 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 6d). 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.

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 III presents the *observed* availability of all experiments calculated as $\frac{writes\_completed}{writes\_tried}$. 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.

Provider	Number of writes	Errors	$\frac{writes\_completed}{writes\_tried}$	wrong hash value
Google	72500	0	100%	0
Amazon	72500	16	99,978%	0
Nirvanix	42000	3	99,993%	0
Rackspace	42000	145	99,655%	5
Box	42000	0	100%	40
HP	42000	0	100%	0

Table III
THE OBSERVED AVAILABILITY DURING THE EXPERIMENT.

IV. Related Work

The main idea underlying our approach is to provide RAID technique at the cloud storage level. In [5] 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-responce protocols for retrievability (POR) [3] and proofs of data possession (PDP) [3] 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 [9] 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 the capabilities such as intelligent file placement based on users' requirements or automatic replication. In addition to it, the RACS system does not try to solve security issues of cloud storage, but focuses more on vendor lock-in. Therefore, the system is not able to detect any data corruption or confidentiality violations.

The future of distributed computing has been a subject of interest for various researchers in recent years. The authors in [7] 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 [4] 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 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.

V. Conclusion

In this paper we focused on the performance evaluation of cloud storage providers in terms of service provider's response time and its resilience (i.e. availability, performance). The latter experiments helped us to understand how cloud storage providers handle high numbers of simultaneous transfers. Using these results we were able to assess the improvement potential of Cloud-RAID performance. Our approach focuses on segmenting the input data into different chunks and transferring them simultaneously. The experiments showed that simultaneous transfers can significantly increase the transmission performance depending on the individual capabilities of cloud providers when dealing with different file sizes and simultaneous transfers.

We concluded that the requirements of implementing our approach 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). Our conducted experiments also provided sufficient results for the identification of an appropriate transfer strategy per provider's capabilities.

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. 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.

VI. future work

Our performance testing revealed that some vendors have optimized 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 provide a strategy to leverage the discoveries for Cloud-RAID optimization. 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. Whilst our system is still under development at present, we have to use the results of the conducted experiment to improve the overall performance and reliability. This includes for instance the predictability and sufficiency of response time and throughput as well as the validation file consistency.

REFERENCES

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

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[3] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. *14th ACM CCS*, 2007.

[4] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 31–46, New York, NY, USA, 2011. ACM.

[5] Kevin D. Bowers, Ari Juels, and Alina Oprea. Hail: A high-availability and integrity layer for cloud storage. *CCS'09*, November 2009.

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

[7] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, August 2008.

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

[9] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. *ACM SOSP*, October 2001.

[10] R. Dingledine, M. Freedman, and D. Molnar. The freehaven project: Distributed anonymous storage service. *The Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[11] Ponemon Institute. Security of cloud computing providers study. online, April, 2011.

[12] Implementation of a Secure and Reliable Storage Above the Untrusted Clouds. Platform for a secure storage-infrastructure in the cloud. *Proceedings of 8th International Conference on Computer Science and Education – ICCSE 2013*, 2013.

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

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

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

[16] Maxim Schnjakin, Michael Goderbauer, Martin Krueger, and Christoph Meinel. Cloud storage and it-security. *Proceedings of the 13th Deutscher IT-Sicherheitskongress (Sicherheit 2013)*, 2013.

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

[18] Maxim Schnjakin and Christoph Meinel. Implementation of cloud-raid: A secure and reliable storage above the clouds. *Proceedings of 8th International Conference on Grid and Pervasive Computing – GPC 2013*, 2013.

[19] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. *IPTPS*, March 2002.