Cost- and Latency-Efficient Redundant Data Storage in the Cloud

Johannes Matt, Philipp Waibel, Stefan Schulte
Distributed Systems Group, TU Wien, Austria
Email: johannes.matt@aon.at, {p.waibel, s.schulte}@infosys.tuwien.ac.at

***Abstract*—With the steady increase of offered cloud storage services, they became a popular alternative to local storage systems. Beside several benefits, the usage of cloud storage services can offer, they have also some downsides like potential vendor lock-in or unavailability. Different pricing models, storage technologies and changing storage requirements are further complicating the selection of the best fitting storage solution.**

In this work, we present a heuristic optimization approach that optimizes the placement of data on cloud-based storage services in a redundant, cost- and latency-efficient way while considering user-defined Quality of Service requirements. The presented approach uses monitored data access patterns to find the best fitting storage solution. Through extensive evaluations, we show that our approach saves up to 30% of the storage cost and reduces the upload and download times by up to 48% and 69% in comparison to a baseline that follows a state-of-the-art approach.

***Index Terms*—Cloud-based storage, Latency-efficient, Cost-efficient, Long-term storage, Data placement optimization**

I. Introduction

The usage of cloud storage services is a popular alternative to the usage of a private storage, e.g., company owned servers, not only for companies but also for private persons and government organizations [1]. Such cloud storage services offer a cost-efficient solution to store data in a highly accessible and reliable way. Especially for smaller and medium-sized enterprises, cloud storage services can help to lower the storage cost, because of the disappearance of the maintenance cost that would occur for a private storage system [2].

Due to the popularity of these storage services, several public cloud storage providers exist, e.g., Google Cloud Storage[1], Microsoft Azure[2] or Amazon S3[3] [3]. Each of them provides its own Web and API interfaces to upload and access data. Each provider offers different storage technologies, pricing schemes, geographical locations and Quality of Service (QoS).

Due to the large amount of different providers and offers, the selection of the cloud storage service that suits a customer's needs best, is not trivial. Several properties have to be taken into account by a customer to find the best-fitting provider. Beside others, the most important properties are: Which QoS level is required, which storage technology should be used, which provider should be avoided, and which geographical location should be preferred.

[1]https://cloud.google.com/storage/
[2]http://azure.microsoft.com/en-us/services/storage/
[3]https://aws.amazon.com/s3/

Also the pricing scheme and, thus, the resulting storage cost have to be taken into account. The provided pricing schemes vary a lot from provider to provider and between the different storage technologies. Furthermore, the data access pattern has to be considered, since it is a significant cost factor [4]. For instance, data which is accessed very seldom, e.g., backups, could be stored on storage services with a cheap price. However, for often used data another cloud storage service with a cheap access price is most likely a better choice. Last but not least, the upload and download latency has to be considered. Especially for frequently used data, a lower access latency is worth striving for, while for seldom used data a higher access latency might be acceptable.

Besides the benefits a cloud storage service can offer, there are also downsides. As a temporary outage in February 2017 of Amazon S3 in Northern Virginia [5] as well as several other examples have shown, even big cloud storage providers struggle with service outages which may lead to temporary data unavailability [6]. Another significant issue is that the selected cloud storage provider could increase the price of the service or go out of business [7]–[9], which leads to the necessity to migrate the data to another cloud storage provider. Such a migration involves administrative and transferring cost. In the worst case, if the provider goes out of business, the data may even be lost as long as no further replication is available. A change of the terms of usage, issued by the cloud provider, or a change of the requirements of the customer to the cloud storage service may also require a migration of the data to another cloud storage provider.

A possible solution to those downsides is the redundant usage of different cloud storage services. Beside the increase of the availability and durability of the data, which comes in hand with the redundant usage of different cloud storage services, this approach decreases the risk of vendor lock-in [10], [11]. In addition, the access latencies can be reduced by using different geographically distributed cloud storage services near the location of the customer [12].

In the work at hand, we address the problem of redundant storage of data, in the following called *data objects*, on cloud storage services while decreasing cost and access latency. For this, we extend our work from [10], where we introduced a cloud storage middleware that stores data objects in a cost-efficient and redundant way. Furthermore, in [10] we formulated a local optimization problem that optimizes the placement of data objects on several cloud storage services while considering user defined requirements, e.g., availability of the data objects. In this work, we present a heuristic optimization approach for the data object placement on multiple cloud storage services that:

- Optimizes the data object placement in a cost-efficient and redundant way.
- Optimizes the data object placement to minimize access latencies.
- Considers predefined requirements, i.e., availability, durability, and vendor lock-in.
- Considers monitored access patterns of the data objects and monitored access latencies of the storage services.

This paper is structured as follows. In Section II, we present required background information for the presented approach, while in Section III, we discuss the general approach of storing data redundantly on cloud storage services. Afterwards, the heuristic optimization approach is discussed in Section IV. Section V discusses the evaluation setup and Section VI presents the evaluation of the novel heuristic approach. The paper then concludes with the discussion of the related work in Section VII and the conclusion in Section VIII.

II. Background

A. Quality of Service

When storing data on cloud storage services, several QoS requirements have to be considered. The most important constraints are availability, durability, vendor lock-in factor and access latency [9], [12]–[14].

1) Durability: The durability describes the probability that data stored on a storage service does not get lost permanently, e.g., due to a hardware failure. This parameter is defined in percentage over a time span, e.g., a durability of 99.99999% over a given year [15].

2) Availability: The availability describes the probability that the permanently stored data can be accessed, i.e., the storage service is up and running, in percentage [15], [16]. For instance, 99.9999% over a given year defines that the data on a storage can be accessed at any time within this year with a probability of 99.9999%.

3) Vendor lock-in factor: A vendor lock-in describes the situation when the data is stored on a specific storage provider and cannot be migrated to another provider [4]. This can happen for example when the provider is temporarily unavailable or goes out of business [6], [11]. The vendor lock-in factor is calculated by $lockin = \frac{1}{N}$, where N is the number of storage services that are used to store a data object ($lockin \in (0, 1]$).

4) Latency: Latency defines the delay between sending a request and receiving a response to the request. It is defined as the end-to-end Round-Trip Time (RTT), where RTT defines the time that elapses between sending a request to a service (here: a storage service) and receiving a response, e.g., the requested data object. The latency is defined as a time period, e.g., a latency of 100ms defines that 100ms after sending a request a response is received [17].

The required QoS attributes are defined in Service Level Agreements (SLAs) between the provider and the customer [16].

B. Erasure Coding

Erasure coding is a redundancy mechanism that splits a data object into several chunks in a way that a subset of those chunks is enough to reconstruct the whole data object. An erasure coding configuration is defined by the tuple (m, n), where m defines the amount of chunks that are required to reconstruct the data object and n ($m < n$) defines the amount of total chunks [18]–[20]. For instance, an erasure coding configuration of (2,3) defines that the data object is split into three chunks in a way that any subset of size two is enough to reconstruct the data object. Thus, erasure coding is a superset of RAID and also of a normal replication system [13], since, e.g., RAID 5 can be achieved by a (4,5) configuration and a replication system by a (1,3) configuration. The latter one will result in three identical replications.

C. Pricing Models

There is great variation of pricing models between the different cloud storage providers and storage technologies. However, most providers use the same foundation for their pricing models: All models charge the used storage space, the outgoing data transfer, and the number of write and read requests on a monthly base. Most providers do not charge for the incoming data transfer and delete operations. Several providers also use a *block rate pricing model* where the price decreases with a higher usage of the service [21], e.g., the more storage is used the cheaper it is to lease additional storage. There are different variants of the block rate pricing model: For example, while Amazon S3 uses the block rate pricing model for the storage cost and for the outgoing data transfer cost, Google Cloud Storage only uses it for the outgoing data transfer cost.

Storage providers with different geographically distributed data centers, called *regions*, are often offering cheaper prices to migrate data between different regions of the same provider. If data has to be migrated from one provider to another provider, the outgoing transfer and the incoming transfer are charged, which is normally more expensive than the migration prices within the same provider.

In addition to different geographical locations, several providers also offer different storage technologies. For instance, Amazon S3 offers the following technologies: Standard Storage, Standard-Infrequent Access (IA) Storage, and Glacier Storage. Each of them with different QoS properties and pricing models, e.g., while the Standard-IA Storage has a cheaper storage price it also has lower availability than the Standard Storage.

Especially in the case of long-term storage services, the pricing models differ from the pricing models of the standard storage services. Long-term storage services are designed for storage of data with rare data access, e.g., backup data. Therefore, those pricing models have lower storage cost but

TABLE I
SLA ATTRIBUTE EXAMPLE, PORTED FROM [10]

File Name	Availability (%)	Durability (%)	Vendor lock-in
image.png	99.8	99.999	0.4
backup.tar	99.9	99.99999	0.5

higher access cost that may include additional data retrieval cost. Long-term storage services often define a *Billing Time Unit* (BTU) and a *Billing Storage Unit* (BSU). The BTU defines a minimum storage duration which is charged as soon as the storage is used. This means that as soon as a data object is stored on such a long-term storage, the whole BTU is charged, no matter if the data object is deleted before the BTU is over. The BSU defines the minimum size of a data object that is billed. If the data object is smaller than the BSU, the BSU is charged, despite the fact that only a part of the storage is used. For instance, the Amazon S3 Standard-IA Storage has a BSU of 128 KB and a BTU of 30 days.

III. Redundant Data Storage in the Cloud

The general idea of a redundant data storage in the cloud is the usage of several cloud storage services to store a data object in a redundant way in order to increase the availability and durability of the data object and to decrease the risk of vendor lock-in. In our former work [10], we presented an approach that uses several cloud storage services to store data objects in a cost-efficient and redundant way. The approach is based on a middleware, called CORA, between the customer and different cloud storage services. The middleware thereby manages the communication between the customer and the cloud storage services and optimizes the placement of the data objects on the storage services.

If a customer uploads a data object, the middleware splits up the data object into several chunks by the usage of erasure coding. Those chunks are then uploaded to different cloud storage services. Each chunk of a data object is stored on a separate storage service, where the amount of storage services is greater or equal to the maximal amount of chunks, to ensure that there are enough chunks left to reconstruct the data object if one storage service is unavailable.

To ensure a cost-efficient storage of the data objects and the fulfillment of the customer-defined SLAs, the middleware uses an optimization algorithm that optimizes the placement of the chunks. The SLA definition is done by the customer for each file. Table I shows an example definition for two data objects. The optimization algorithm uses monitored access pattern information of the chunks to optimize the placement of them. For instance, if a chunk is accessed very seldom, the optimization will select a long-term storage service to store the chunk. If the cost can be decreased by migrating one or more chunks, this will be done automatically by the middleware.

If a customer wants to download a data object, the middleware downloads the related chunks from the storage services and recreates the original data object, which is then sent back to the customer. To save cost, the middleware only downloads as many chunks as required to reconstruct the data object. For instance, for an erasure coding configuration of (2,3), only two chunks have to be downloaded to reconstruct the data object.

Furthermore, if the middleware detects the unavailability of a storage service, all chunks that are stored on this particular service are recreated and stored on another available storage service. This recreation is done by using the remaining available chunks. After the chunk is recreated, the optimization is used to find the best-fitting storage service for the recreated chunk to guarantee the fulfillment of the defined SLAs and to find a cost-efficient placement solution once again.

IV. Data Object Placement

In the following, a heuristic approach for the optimization of the data object placement on multiple cloud-based storage services is introduced.

Since the data object placement on multiple cloud-based storage services is an NP-complete problem [14], [22], we present in the work at hand a heuristic approach that aims at finding a cost- and latency-efficient solution in a reasonable amount of time. To decrease the cost, the algorithm uses monitored historical access pattern information of the data objects and for the access latency optimization the algorithm uses monitored historic access latency information.

The basic approach of the heuristic is built on two observations [23]:

- The biggest cost saving is achieved by migrating not or rarely used chunks from standard storage services to long-term storage services.
- For an erasure coding configuration (m, n), only m chunks have to be downloaded to reconstruct a data object. Therefore, by storing m chunks on standard storage services and the remaining $n - m$ chunks on long-term storage services, a cost saving can be achieved since the storage prices of long-term storage services are lower than the storage prices of standard storage services. However, long-term storage services have a higher download price in comparison to standard storage services. By choosing the cloud storage services with the cheapest download price for the m chunks, further cost savings can be achieved.

Those two observations are the basis for the heuristic approach for the placement optimization of the chunks. The heuristic is based on two algorithms, the first one is triggered by an upload of a data object (Algorithm 1) and the second one by a download of a data object (Algorithm 2). Both algorithms use ranking functions that create rankings of the storage services to find the best-fitting storage solution. The ranking of the storage services is done according to their pricing model and their access latency.

In the following, the ranking function as well as the algorithms will be explained in detail.

A. Ranking Function

In eq. (1) the cost ranking function, which creates a ranking value of a storage service according to its pricing model, is shown. Furthermore, eq. (2) shows the ranking function that considers the pricing model and the monitored latency.

The result of eq. (1) is a ranking value of a storage service s by considering a chunk f. The idea of the ranking function is to calculate a ranking value according to the cost that would be charged if a data object was stored on a storage service for one hour and downloaded once. In the equation, σ_f returns the size of a chunk f. Furthermore, p_s^S returns the storage price of s, p_s^W and p_s^R return the write and read request prices of s, $p_s^{T_{out}}$ returns the outgoing data transfer price, and p_s^{ret} the data retrieval price. Finally, $h_f \in \{0,1\}$ indicates if f is stored on a long-term storage service or not. If f is stored on a long-term storage service $h_f = 1$, otherwise $h_f = 0$.

$$r_{s,f} = \sigma_f \cdot \frac{p_s^S}{30 \cdot 24} + p_s^W + p_s^R + \sigma_f \cdot (p_s^{T_{out}} + p_s^{ret} \cdot h_f) \quad (1)$$

Equation (1) is composed of four subterms. The first subterm $\sigma_f \cdot \frac{p_s^S}{30 \cdot 24}$ calculates the price of storing a chunk f on a cloud storage service s for one hour. Since the storage prices are given on a monthly basis the price has to be divided by $30 \cdot 24$. Subsequently, p_s^W and p_s^R add the cost of one write and one read request to the calculation. Finally, the last subterm $\sigma_f \cdot (p_s^{T_{out}} + p_s^{ret} \cdot h_f)$ calculates the transfer cost for downloading the data object once. To facilitate the calculation of the ranking function to get a highly scalable solution, we neglect the block rate pricing model and always take the first step in the block rate pricing model.

In eq. (2), the cost ranking function is extended to also consider the latency for each storage service.

$$r_{s,f}^L = r_{s,f} \cdot (1 + \alpha_{s,f}) \quad (2)$$

The equation uses the already discussed cost ranking function $r_{s,f}$ from eq. (1) and multiplies it with $1 + \alpha_{s,f}$. The subterm $\alpha_{s,f} \in [0,1]$ is a weighted multiplier that returns high values for storage services with a high latency and low values for storage services with a low latency. Therefore, the resulting value will increase the result of $r_{s,f}$ significantly for storage services with a high latency and increase it slightly if the service has a low latency.

B. Placement Algorithms

After discussing the ranking functions, the upload and download algorithms can be explained. Algorithm 1 shows the (pseudo-) code that is triggered each time an upload of a data object takes place. This algorithm takes care of finding the best-fitting storage services, according to the described ranking functions, for a new data object, respectively its chunks. Which ranking function is used depends on the requirements of the customer. If the data object placement should be optimized only by considering the cost, eq. (1) is used, if the latency also has to be considered, eq. (2) is used.

Since the access pattern is not known at the beginning, the best-fitting storage services are the cheapest standard storage services. However, as explained above by using an erasure coding configuration, m chunks are enough to reconstruct the data object. Algorithm 1 makes use of this characteristic and uploads in the beginning m chunks to standard storage services and the remaining $n-m$ chunks to long-term storage services. The download algorithm will then only download the m chunks on the standard storage services as long as all of them are available. This approach helps us to decrease the cost already at the beginning, since most providers do not charge for incoming traffic.

In Algorithm 1, lines 3 and 4 use the ranking function to get a list of ranked standard and long-term storage services. In lines 5-7, the m best-fitting standard storage services are selected and stored in the set $storages$. In lines 8-10, the $n-m$ best-fitting long-term storage services are selected and also stored in the set $storages$. Subsequently, in lines 11 and 12, all n chunks are assigned to the set $placement$. This results in an assignment of m chunks of the data object f to the standard storage services and the remaining $n-m$ chunks assigned to the long-term storage services.

Algorithm 1 Upload Placement Function
Input: The data object f that should be uploaded
Output: A list of chunk to storage service assignments

```
1:  $placement \leftarrow \emptyset$
2:  $storages \leftarrow \emptyset$
3:  $standardStorages \leftarrow rankedStandardStorages(f)$
4:  $longTermStorages \leftarrow rankedLongTermStorages(f)$
5:  for $i \leftarrow 1$ to $m$ do
6:      $storages.add(standardStorages[i])$
7:  end for
8:  for $i \leftarrow 1$ to $n-m$ do
9:      $storages.add(longTermStorages[i])$
10: end for
11: for $i \leftarrow 1$ to $n$ do
12:     $placement.add(storages[i], f.chunks[i])$
13: end for
14: return $placement$
```

Algorithm 2 presents the algorithm that is triggered by the download of a data object. For a cost-efficient solution, our heuristic aims at optimizing the placement of unused data objects. The algorithm considers a data object as unused if it has not been downloaded for a predefined amount of time. The algorithm holds a list of all unused data objects in predefined time periods. After each time period, the placement of all unused data objects is optimized. Furthermore, to reduce the complexity of the algorithm, the update of the list of unused data objects is not done after each download but after predefined intervals.

In Algorithm 2, lines 2 and 3 are checking if the time between the last update of the unused data object list and the current time is greater than or equal to the predefined step interval called $stepInterval$. Following, lines 4-9 are checking, for all data objects, if they are used or unused. If they are unused, they are stored in the set $unused$. The resulting set of unused data objects of this iteration is then added to the set $allUnusedObjects$ in line 10. Furthermore, the variables $timePeriodCount$ and $previousTime$ are updated in lines

11 and 12. If $timePeriodCount \geq timePeriodThreshold$ is reached, an optimization of all unused data objects takes place, shown in line 14.

Lines 15-23 iterate through all unused data objects stored in $allUnusedObjects$. Following with an iteration of all chunks of the currently unused data object, shown in lines 16 and 17. Lines 18-21 are then checking if the current chunk is already stored on a long-term storage service or not. If it is not stored on a long-term storage service, the placement of the chunk is changed to a long-term storage service. Similar as in Algorithm 1, the best-fitting long-term storage service is determined by the ranking function. Finally, $timePeriodCount$ and $allUnusedObjects$ are reset (lines 24 and 25) and the new placement is returned (line 27).

Algorithm 2 Download Placement Function
Input: All data objects F
Output: A list of chunk to storage service assignments for all unused data objects

```
 1: $placement \leftarrow \emptyset$
 2: $timeDiff \leftarrow currentTime - previousTime$
 3: **if** $timeDiff \geq stepInterval$ **then**
 4:     $unused \leftarrow \emptyset$
 5:     **for all** $f \in F$ **do**
 6:         **if** $isUnused(f)$ **then**
 7:             $unused.add(f)$
 8:         **end if**
 9:     **end for**
10:     $allUnusedObjects.put(timePeriodCount, unused)$
11:     $timePeriodCount \leftarrow timePeriodCount + 1$
12:     $previousTime \leftarrow currentTime$
13: **end if**
14: **if** $timePeriodCount \geq timePeriodThreshold$ **then**
15:     **for all** *unused f in all past time periods* **do**
16:         **for** $i \leftarrow 1$ **to** $n$ **do**
17:             $chunk \leftarrow f.chunks[i]$
18:             **if** *currentStorage(chunk) is not long-term* **then**
19:                 $stor \leftarrow bestRankedLongTerm(chunk)$
20:                 $placement.add(stor, chunk)$
21:             **end if**
22:         **end for**
23:     **end for**
24:     $timePeriodCount \leftarrow 0$
25:     $allUnusedObjects \leftarrow \emptyset$
26: **end if**
27: **return** $placement$
```

The computational time of the update algorithm (Algorithm 1) mainly depends on the amount of chunks and the amount of standard and long-term storage services. Since our algorithm requires that the amount of storage services S is $|S| \geq n$, where n is the total amount of chunks of a data object, we know that $rankedStandardStorages(f)$ and $rankedLongTermStorages(f)$ together need equally or more iterations than lines 11-13 in Algorithm 1. This results in a complexity of $O(|S|)$. Since the amount of storage services has a small upper limit, also the complexity of the algorithm is small and does not increase if the amount of data objects increases.

The complexity of the download algorithm (Algorithm 2) mainly depends on lines 6-8 and lines 15-23. Lines 6-8 iterate through all stored data objects and check if they are unused. Lines 15-23 then iterate through all unused data objects of the last $timePeriodThreshold$. The resulting complexity is therefore $O(max(|\textit{data objects in allUnusedObjects}|, |F|)$.

TABLE II
EVALUATION STORAGE SERVICES

Provider	Region	Storage Class
AWS S3	US N. Virginia	Standard Storage
AWS S3	US N. Virginia	Infrequent Access (IA)
AWS S3	US N. California	Standard Storage
AWS S3	EU Frankfurt	Standard Storage
AWS S3	EU Frankfurt	Infrequent Access (IA)
AWS S3	Asia Pacific Tokyo	Standard Storage
AWS S3	Sao Paulo	Standard Storage
Google Cloud	Europe	Standard Storage
self-hosted	-	Standard Storage
self-hosted	-	Long-term Storage

V. Evaluation Setup

In the following we discuss the evaluation setup, before we discuss the actual evaluation of the presented approach.

A. Prototype

For the evaluation, we extended the middleware CORA, presented in our former work [10], with the heuristic optimization approach presented in the work at hand. The prototype triggers the heuristic optimization each time a data object is accessed.

B. Storage Services

For the evaluation, we use several real-world cloud storage systems. Table II provides an overview of the used cloud storage services, their providers and the chosen storage technology.

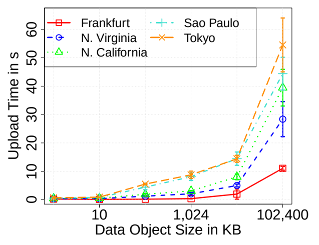
For the evaluation, we use the real-world pricing models from AWS S3[4] and Google Cloud Storage[5]. For the self-hosted Standard Storage, the pricing model of AWS S3 Frankfurt Standard Storage is used and for the self-hosted Long-term Storage, the one from AWS S3 Frankfurt IA is used.

C. Evaluation Data

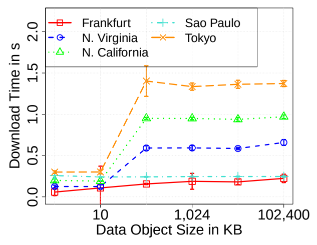
For the evaluation, we use a real-world access trace presented in [24]. This access trace contains data object' access traces on a cloud storage used by more than 1,000,000 users over a time period of 30 days.

For the evaluation, we extracted a dataset with 100,000 randomly selected data objects and one dataset with around 1,900 data objects. The data object usage of the larger dataset is as follows: 0.12% of the data objects are used more than 100 times over the 30 days, 0.29% are used more than 30 times

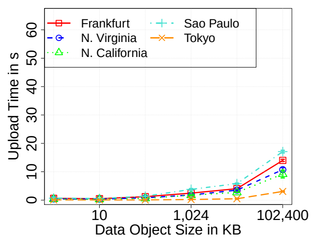
[4]https://aws.amazon.com/s3/pricing/
[5]https://cloud.google.com/storage/pricing



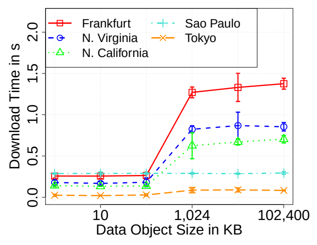
(a) Upload times location Vienna



(b) Download times location Vienna



(c) Upload times location Tokyo



(d) Download times location Tokyo

Fig. 1. Measured upload and download times from Vienna and Tokyo to different AWS S3 cloud storage service regions.

and less than 100 times during this time and the remaining objects are used less than 30 times. The data object usage of the smaller dataset is as follows: 31% of the data objects are used more than 100 times, 0.7% are used more than 30 times and less than 100 times and the remaining objects are used less than 30 times.

Beside the normal data object' access, the trace also contains three DDOS attacks. Since we are only considering normal usage of a cloud storage service, we did not include data objects that are used during those attacks.

As SLAs for the data objects, we define for each of them a durability of 99.9999999%, an availability of 99.99% and a vendor lock-in factor of 0.5.

D. Cloud Service Upload and Download Times Measurement

To be able to evaluate the latency consideration functionality of our optimization approach together with the dataset discussed above, we measured different download and upload times upfront. During the evaluation we use these measured times to simulate the required upload and download of the chunks. For a better analysis of the approach, we measured the upload and download times from two different geographically distributed locations, Vienna and Tokyo, to different AWS S3 cloud storage service regions around the world. For the measurement, we issued several read and write requests of differently sized data objects and measured the total execution time for every request. Figure 1 shows the measured results from Vienna and Tokyo to the different cloud storage services. We repeated these measurements 12 times at different times of the day and week. In the evaluation, we use the average upload and download time for each storage region. For the Google Cloud storage service, we use the measured times for AWS S3 Frankfurt. The same is also used for the self-hosted storage service except for the case when the evaluation is done from Vienna. In this case we set the times to nearly 0.

E. Evaluation Process

In each evaluation, we iterate through the dataset and perform the recorded operations (upload, download, and update). Furthermore, we use the entire 30 days for each evaluation and the full storage service set presented in Section V-B. The BTUs of the long-term storage services and the billing period of all storage services are set to one week. The history time step interval is set to 12 hours. Each history time step contains the amount of access operations in this time period. This setting already provided good results in our previous works [10] and [23]. The variable $timePeriodThreshold$ is set to 10, since this value provided the best results in different testing scenarios.

To analyze the performance of our optimization approach, in terms of time and memory consumption, we log for each optimization the required duration and the metadata size.

F. Baseline

For the baseline, we use the same dataset but disable the optimization. Instead, a fixed storage service set that is equally used to store the data objects, is utilized. This simulates the usage of the cloud storage services without an optimization. The fixed storage service set contains the cheapest three standard storage services: AWS S3 US Northern Virginia, AWS S3 EU Frankfurt and the self-hosted.

VI. Evaluation Scenarios

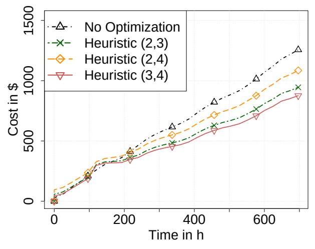
In the following, we evaluate our approach by evaluating the cost optimization approach in Section VI-A, and the latency consideration optimization in Section VI-B.

A. Cost Optimization Evaluation

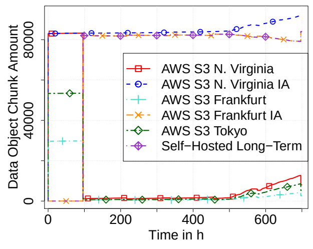
In this evaluation scenario, we evaluate the behavior of our approach with a large amount of data objects against the baseline. This evaluation scenario uses the dataset with 100,000 data objects. We evaluate the behavior with different erasure coding configurations, namely (2,3), (2,4), and (3,4).

Evaluation Hypothesis: At the beginning of the evaluation, the heuristic uploads all chunks of the data objects to the best-ranked standard storage services and the best-ranked long-term storage services, according to the ranking function from Section IV-A. How many chunks are stored on standard storage services and how many on long-term storage services is defined by the erasure coding configuration, as described in Section IV-B. The upload to long-term storage services will immediately increase the cost, due to the accrued BTU cost.

After the initial upload, the cost of the heuristic optimization rises slower than the baseline cost, due to the already charged BTU cost, which has the consequence that no additional storage cost are charged. After a while the heuristic optimization will migrate unused chunks from standard to long-term storage services. This will increase the cost due to the BTU and the migration. However, as a consequence the heuristic optimization cost will rise slower than the baseline. After some time, the cost of the heuristic optimization will be lower than the baseline and will stay lower.



(a) Cumulative cost



(b) Data object chunk distribution

Fig. 2. Cost optimization evaluation results.

Evaluation Execution: Figure 2a shows the cumulative cost of the heuristic optimizations and the baseline. Figure 2b shows the chunk distribution on the different storage services during the (2,3) erasure coding configuration evaluation.

As shown in Figure 2a, at the beginning of the evaluation all three heuristic optimization evaluations have higher cost than the baseline due to the additional BTU cost of the long-term storage services. Furthermore, it can be observed that the erasure coding configuration (2,4) has the highest cost and the configuration (3,4) the lowest cost at the beginning. This is due to the fact that the (2,4) configuration uploads two chunks to long-term storage services, which charge twice as many on BTU cost than for the (3,4) configuration where only one chunk is uploaded to a long-term storage service. Besides, it can be observed that the evaluation with a (2,3) configuration has a higher initial cost than the one with a (3,4) configuration. This is because the (2,3) configuration results in bigger chunks than the (3,4) configuration and, therefore, higher BTU cost are charged. This chunk distribution can also be seen in Figure 2b where the chunks are stored on standard storage services (e.g., AWS S3 Frankfurt) as well as on long-term storage services (e.g., AWS S3 Northern Virginia IA).

After the initial upload, it can be seen in Figure 2a that the cost of the heuristic optimization evaluations rise slower than the baseline, since no additional cost for the long-term storage services are charged as long as the BTU is not over. It can also be seen that after some time the (3,4) and (2,3) configuration already achieve a cost saving in comparison to the baseline.

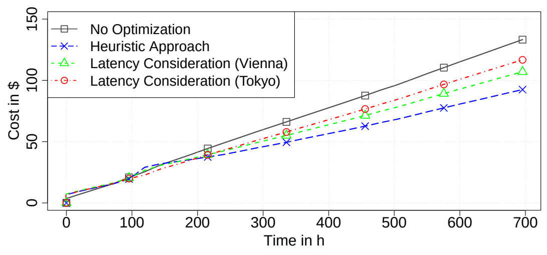
After 100 h, the heuristic approach migrates all unused data objects from standard to long-term storage services. This again increases the cost for the heuristic optimization evaluations due to the additional BTU cost. However, this results in a slower rise of the cost in comparison to the baseline. The migration can also be observed in Figure 2b.

Finally, in Figure 2a it can be seen that after 210 h all three heuristic evaluations have lower cost than the baseline and stay lower for the rest of the evaluation.

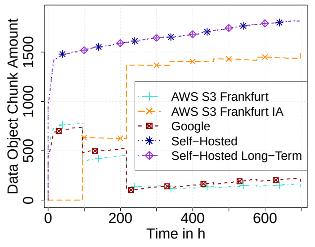
In summary, we are able to save 25% of cumulative cost with a (2,3) erasure coding configuration, 14% with a (2,4) configuration, and 30% with a (3,4) configuration.

B. Latency Optimization Evaluation

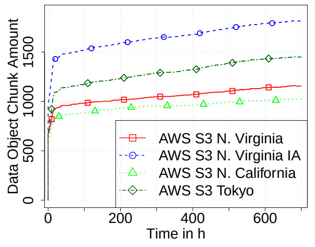
In this evaluation, we analyze the latency consideration behavior of our optimization approach. For this, we execute three different evaluations and compare them to the baseline.



(a) Data object chunk distribution (location Vienna).



(b) Data object chunk distribution (location Vienna).



(c) Data object chunk distribution (location Tokyo).

Fig. 3. Latency optimization evaluation results.

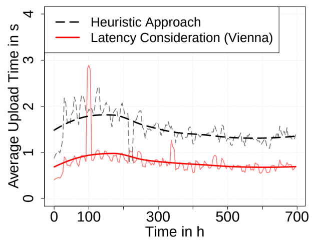
In the first evaluation, we evaluate our approach based on the location of Vienna and in the second we evaluate the approach based on the location of Tokyo. Furthermore, the third evaluation run is done with only cost optimization and deactivated latency optimization. This evaluation uses the smaller dataset and as an erasure coding configuration we use a (2,3) configuration for all evaluations.

Evaluation Hypothesis: Similar to the cost optimization evaluation (Section VI-A), at the beginning of the evaluation the cost will be higher than for the baseline due to the storage of chunks on long-term storage services. However, due to the different locations of the evaluation (Vienna and Tokyo) the optimization chooses different storage locations, e.g., for the evaluation from Vienna it is expected that the self-hosted, the AWS S3 Frankfurt and the Google storage service are chosen since those storage services are all near Vienna.

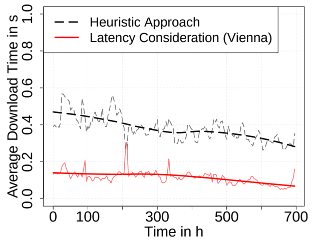
Similar to the cost optimization evaluation, after some time the heuristic optimization evaluation will migrate unused chunks to long-term storage services, which will result in a cost saving in comparison to the baseline after some time.

Evaluation Execution: Figure 3a shows the cumulative cost of the heuristic optimization evaluation with latency consideration, of the heuristic optimization evaluation without latency consideration and of the baseline. Figure 3b shows the chunk distribution on the different storage services during the latency consideration evaluation from Vienna and Figure 3c shows the chunk distribution during the evaluation from Tokyo.

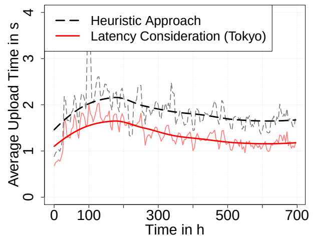
In Figure 3a, a cost increase can be observed already at the beginning of the evaluation due to the BTU of the used long-term storage services. However, as long as the BTU is not over, no additional cost for those chunks are charged and, therefore, the cost of the heuristic approaches rise slower than for the baseline. This chunk upload can also be observed in Figure 3b and Figure 3c.



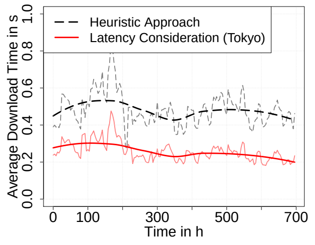
(a) Upload times location Vienna



(b) Download times location Vienna



(c) Upload times location Tokyo



(d) Download times location Tokyo

Fig. 4. Upload and download times of the latency consideration approach based on the location of Vienna and Tokyo.

After 100 h, it can be observed in Figure 3a and in Figure 3b that a migration of unused chunks from standard to long-term storage services, in case of the evaluation from Vienna, takes place. Also during the evaluation without latency consideration, a migration of unused chunks can be observed in Figure 3a. However, as can be seen in Figure 3c, during the evaluation from Tokyo no chunks are migrated. This is due to the combination of cost and latency consideration of the optimization approach. The best-ranked long-term storage service in respect of cost and latency is AWS S3 Northern Virginia IA. All other long-term storage services have a higher latency and the cost saving is not big enough to compensate this. Since all data objects already have a chunk stored on AWS S3 Northern Virginia IA, no more chunks are migrated to this storage service to not violate the vendor lock-in factor defined in the SLAs.

After 150 h, it can be seen in Figure 3a that from this point in time all heuristic evaluations have a lower cost than the baseline. In Figure 3b, it can be further observed that a second migration of unused chunks from standard to long-term storage services takes place after 220 h.

In summary, we are able to save 12% of the cumulative cost with the latency consideration evaluation based on Tokyo, in comparison to the baseline, and 20% with the latency consideration evaluation based on Vienna. Compared to the heuristic without latency consideration, the latency consideration evaluation based on Vienna has a cost increase of 11% and the one based on Tokyo an increase of 19%.

Figure 4 shows the upload and download times of the data objects for the latency consideration evaluations based on Vienna and Tokyo in comparison to the execution without latency consideration. As can be seen in both graphs, the upload and download times of the latency consideration evaluations are lower than for the heuristic evaluation without latency consideration. In case of the execution from location Vienna, the average upload time is 48% faster and the average download time is 69% faster than for the evaluation without latency consideration. For the evaluation with location in Tokyo, the average upload time is 26% faster and the average download time is 46% faster in comparison to the execution without latency consideration.

C. Performance Assessment

Duration: Table III shows the average optimization durations of all evaluations. As can be seen, our heuristic approach needs on average 4.8ms for the optimization of the data object placement during the cost optimization evaluation from Section VI-A. In case of the latency consideration evaluation from Section VI-B, it can be seen that without latency consideration the duration is not noticeable. For the evaluation with latency consideration a slight increase can be observed. However, with an average duration of 0.41ms it is negligible.

Metadata: Our heuristic approach uses the monitored access information of the last BTU of each chunk. For the cost optimization evaluation from Section VI-A with the (2,3) configuration the size of this metadata aggregates to 865MB and for the (2,4) and (3,4) configuration to 1.18GB. The additional size in the latter case is due to the additional chunk for each data object. For the latency consideration evaluation the metadata aggregates to 16MB.

VII. Related Work

In our previous work, we formulated the data object placement problem for a local optimization [10] and a global optimization approach [23]. In addition, we present a global optimization heuristic based on data object classification in [23]. In comparison to our previous work, the heuristic approach presented in this work aims at a high scalability with short optimization durations that can be used for a large amount of data objects. Furthermore, the optimization approaches in [10] and [23] do not include latency considerations.

Besides our own work [10], [23], the work of Papaioannou et al. [14], Abu-Libdeh et al. [9], Zhang et al. [4] and Bermbach et al. [25] are worth mentioning. For a detailed description of them, we refer to [10] and [23].

Despite the importance of latency for cloud storage services, relatively little research has been done in this field. In this respect the work of Wu et al. [26] has to be mentioned. The authors present a storage system, called SPANStore that offers a cost-efficient storage solution on multiple cloud storage services that also considers latencies. In comparison to our own work, SPANStore uses full replication instead of erasure coding, which increases the needed storage drastically. Furthermore, SPANStore does not take long-term storage services and possible block rate pricing models into account.

VIII. Conclusion

Using multiple cloud storage services to store data objects in a redundant way is an obvious choice in order to avoid vendor lock-in and to get a high availability and durability of the

TABLE III
AVERAGE OPTIMIZATION DURATIONS IN MILLISECONDS (STANDARD DEVIATION)

Days	Cost Optimization Evaluation			Latency Optimization Evaluation		
	(2,3)	(2,4)	(3,4)	Without Latency	Vienna	Tokyo
1 - 5	1.38 (σ = 1.74)	1.38 (σ = 1.91)	1.44 (σ = 1.73)	0.09 (σ = 1.50)	1.09 (σ = 5.04)	1.19 (σ = 5.78)
6 - 10	5.47 (σ = 149.21)	5.63 (σ = 153.35)	5.72 (σ = 176.21)	0.02 (σ = 0.83)	0.46 (σ = 46.77)	0.31 (σ = 32.76)
11 - 15	4.59 (σ = 23.46)	4.41 (σ = 2.15)	4.37 (σ = 2.75)	0.01 (σ = 0.29)	0.37 (σ = 41.77)	0.33 (σ = 47.23)
16 - 20	4.71 (σ = 2.08)	4.37 (σ = 2.17)	4.22 (σ = 2.29)	0.02 (σ = 0.38)	0.10 (σ = 3.12)	0.25 (σ = 42.22)
21 - 25	4.69 (σ = 4.35)	4.55 (σ = 2.52)	4.47 (σ = 2.44)	0.02 (σ = 0.40)	0.09 (σ = 4.02)	0.14 (σ = 30.62)
26 - 30	5.83 (σ = 27.41)	5.89 (σ = 22.22)	5.80 (σ = 7.54)	0.02 (σ = 0.45)	0.05 (σ = 3.47)	0.15 (σ = 30.74)

data. Within this work, we presented a heuristic optimization approach offering a high-performance data object placement optimization on multiple cloud storage services for a cost- and latency-efficient redundant storage of the data. Moreover, the approach guarantees the fulfillment of user-defined SLAs, based on a ranking of the available storage services according to their cost and upload and download latency.

We evaluated the presented optimization approach and compared the result with a placement solution without an optimization but a fixed cloud storage service set. We showed that our solution decreases the storage cost by up to 30%, as well as the upload and download times by up to 48% and 69%, in comparison to the baseline.

In our future work, we plan to extend our approach so that it considers privacy aspects by using hybrid clouds. Furthermore, to be able to react more dynamically to different scenarios, we will extend our algorithm so that it can dynamically adapt its configuration parameters during runtime. We plan also to evaluate our approach in different real world scenarios, e.g., big data processing in the domain of smart factories.

ACKNOWLEDGMENT

This work is partially supported by the Commission of the European Union within the CREMA H2020-RIA project (Grant agreement no. 637066).

REFERENCES

[1] E. Allen and C. M. Morris, "Library of congress and duracloud launch pilot program using cloud technologies to test perpetual access to digital content," in *Library of Congress, News Release*, July 14 2009.

[2] P. Gupta, A. Seetharaman, and J. R. Raj, "The usage and adoption of cloud computing by small and medium businesses," *Int. Journal of Information Management*, vol. 33, no. 5, pp. 861–874, 2013.

[3] B. Butler, "Gartner: Top 10 cloud storage providers," http://www.networkworld.com/article/2162466/cloud-computing/cloud-computing-gartner-top-10-cloud-storage-providers.html, Accessed: Sep. 2017.

[4] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "CHARM: A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability," *IEEE Trans. on Cloud Comp.*, vol. 3, no. 3, pp. 372–386, 2015.

[5] "Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region," https://aws.amazon.com/message/41926/, Accessed: March 2017.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," *Comm. of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[7] D. Bermbach, T. Kurze, and S. Tai, "Cloud federation: Effects of federated compute resources on quality of service and cost," in *2013 IEEE Int. Conf. on Cloud Engineering*, 2013, pp. 31–37.

[8] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, and S. Dustdar, "Winds of change: From vendor lock-in to the meta cloud," *IEEE Internet Comp.*, no. 1, pp. 69–73, 2013.

[9] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A case for cloud storage diversity," in *1st ACM Symp. on Cloud Comp.*, 2010, pp. 229–240.

[10] P. Waibel, C. Hochreiner, and S. Schulte, "Cost-efficient data redundancy in the cloud," in *9th Int. Conf. on Service-Oriented Comp. and Applications*, 2016, pp. 1–9.

[11] G. Vernik, A. Shulman-Peleg, S. Dippl, C. Formisano, M. C. Jaeger, E. K. Kolodner, and M. Villari, "Data on-boarding in federated storage clouds," in *6th Int. Conf. on Cloud Comp.*, 2013, pp. 244–251.

[12] Y. F. R. Chen, "The Growing Pains of Cloud Storage," *IEEE Internet Comp.*, vol. 19, no. 1, pp. 4–7, Jan 2015.

[13] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Revised Papers from the First Int. Workshop on P2P Systems*, 2002, pp. 328–338.

[14] T. G. Papaioannou, N. Bonvin, and K. Aberer, "Scalia: An adaptive scheme for efficient multi-cloud storage," in *Int. Conf. on High Performance Comp., Networking, Storage and Analysis*, 2012, pp. 20:1–20:10.

[15] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in *2008 Int. Workshop on Data-aware Distributed Computing*, 2008, pp. 55–64.

[16] M. Alhamad, T. Dillon, and E. Chang, "Conceptual SLA framework for cloud computing," in *4th IEEE Int. Conf. on Digital Ecosystems and Technologies*, 2010, pp. 606–610.

[17] C. Pei, Y. Zhao, G. Chen, R. Tang, Y. Meng, M. Ma, K. Ling, and D. Pei, "Wifi can be the weakest link of round trip network latency in the wild," in *The 35th Annual IEEE Int. Conf. on Comp. Communications*, 2016, pp. 1–9.

[18] J. S. Plank, "Erasure codes for storage systems: A brief primer," *Login: The USENIX Magzine*, pp. 44–50, 2013.

[19] M. Schnjakin, T. Metzke, and C. Meinel, "Applying erasure codes for fault tolerance in cloud-raid," in *2013 IEEE 16th Int. Conf. on Computational Science and Engineering*, 2013, pp. 66–75.

[20] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *4th Int. Conf. on P2P Systems*, 2005, pp. 226–239.

[21] M. Naldi and L. Mastroeni, "Cloud storage pricing: a comparison of current practices," in *2013 Int. Workshop on Hot Topics in Cloud Services*, 2013, pp. 27–34.

[22] C.-W. Chang, P. Liu, and J.-J. Wu, "Probability-based cloud storage providers selection algorithms with maximum availability," in *2012 41st Int. Conf. on Parallel Processing*, 2012, pp. 199–208.

[23] P. Waibel, J. Matt, C. Hochreiner, O. Skarlat, R. Hans, and S. Schulte, "Cost-optimized Redundant Data Storage in the Cloud," *Service Oriented Comp. and Applications*, vol. NN, pp. NN–NN, 2017.

[24] R. Gracia-Tinedo, Y. Tian, J. Sampé, H. Harkous, J. Lenton, P. García-López, M. Sánchez-Artigas, and M. Vukolic, "Dissecting UbuntuOne: Autopsy of a Global-Scale Personal Cloud Back-end," in *2015 Internet Measurement Conference*, 2015, pp. 155–168.

[25] D. Bermbach, M. Klems, S. Tai, and M. Menzel, "MetaStorage: A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs," in *IEEE Int. Conf. on Cloud Comp.*, 2011, pp. 452–459.

[26] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "SPANStore: Cost-effective Geo-replicated Storage Spanning Multiple Cloud Services," in *24th ACM Symp. on Operating Systems Principles*, 2013, pp. 292–308.