Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Dynamic Optimization of Data Object Placement in the Cloud

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Johannes Matt
Matrikelnummer 1126295

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.-Ing. Stefan Schulte
Mitwirkung: Dipl.-Ing. Philipp Waibel

Wien, 21. April 2017

Johannes Matt	Stefan Schulte

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Dynamic Optimization of Data Object Placement in the Cloud

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Johannes Matt
Registration Number 1126295

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.-Ing. Stefan Schulte
Assistance: Dipl.-Ing. Philipp Waibel

Vienna, 21st April, 2017

Johannes Matt	Stefan Schulte

Erklärung zur Verfassung der Arbeit

Johannes Matt
Lerchenfelder Straße 46/4, 1080 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. April 2017

Johannes Matt

Acknowledgements

Hereby I would like to thank everyone who supported me during the work of my thesis.

First, I would like to express my sincere gratitude to my advisors Stefan Schulte and Philipp Waibel for their continuous support and their invaluable feedback. The numerous discussions and their constructive feedback have essentially contributed to the success of this thesis. Especially the possibility to work in an office of the Distributed Systems Group was a great opportunity for myself as this allowed me to maintain close contact with my advisors. Thereby, I want to thank everyone of the DSG for the pleasant working environment. Also, I want to thank Olena Skarlat for the help in improving the optimization model with her profound understanding of computational problems.

I would also like to thank my friends for supporting me and making sure that there is always space for some fun. Together we went through countless unforgettable experiences throughout the course of our studies.

Finally, I would like to express my deepest gratitude to my family for providing me with relentless support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

Kurzfassung

In den letzten Jahren sind Cloud-basierte Speicherlösungen zu einer weit verbreiteten Alternative zu lokalen Speichersystemen geworden. Benutzer von Cloud-basierten Speichersystemen können von einigen Vorteilen profitieren. Einige dieser Vorteile sind die bessere Verfügbarkeit, erweiterte Langlebigkeit und die geringeren IT-Administrationskosten. Allerdings gibt es auch einige Nachteile, die bei der Verwendung von Cloud-basierten Speichern berücksichtigt werden müssen, beispielsweise das Vendor-Lock-In Problem und die mögliche Nichtverfügbarkeit der Daten. Um diese Probleme zu vermeiden, entwerfen wir in dieser Arbeit ein Systemmodell, das mehrere Cloud-Speicher für das Speichern der Daten verwendet. Durch die Verwendung dieses Modells ergibt sich die Möglichkeit, Daten redundant zu speichern und die günstigste Speicherlösung zu finden.

In dieser Arbeit formulieren wir ein globales Optimierungsproblem, welches historische Zugangsinformationen über die Daten in Betracht zieht und dabei vorgegebene Dienstgüteanforderungen erfüllt, um eine kosteneffiziente Speicherlösung zu finden. Außerdem stellen wir einen hochskalierbaren heuristischen Ansatz vor, mit dem große Datenmengen verarbeitet werden können. Die Heuristik ermöglicht es, eine Speicherlösung zu finden, die im Hinblick auf die Kosteneffizienz nahe an die optimale Lösung der globalen Optimierung herankommt. Weiters beschreiben wir einen Ansatz, welcher Latenzen in Betracht zieht. Dieser Ansatz berücksichtigt die Latenzen der verschiedenen Cloud-Speicher bei der Optimierung der Speicherlösung. Dabei wird das Ziel verfolgt, zusätzlich zu einer Kostenreduktion auch möglichst geringe Latenzen zu erreichen, was schlussendlich zu einer besseren Benutzererfahrung führt.

Wir evaluieren alle drei Optimierungsansätze gründlich und zeigen damit die Vorteile der entwickelten Ansätze auf. Für die Evaluierungen vergleichen wir unsere Optimierungsansätze mit einem gängigen Standardansatz. Wir erklären die korrekte Funktionsweise der Optimierungsansätze in einer detaillierten Analyse der Resultate. Dabei zeigen wir, dass wir mehr als 35% der gesamten Speicherkosten im Vergleich zu dem gängigen Standardansatz einsparen können.

Abstract

The use of cloud-based storages to store data has become a popular alternative to traditional local storage systems. Users of cloud-based storages can benefit from a lot of advantages, such as higher data availability, extended durability and lower IT administration cost. However, there also exist drawbacks in using cloud-based storage systems. Among the biggest drawbacks are the problem of vendor lock-in and possible unavailability of the data. To overcome these problems, in this thesis we formulate a system model that makes use of multiple cloud storages to store data. The usage of this system model allows the redundant storage of data and aims at finding the cheapest possible storage solution.

In this thesis we formulate a global optimization problem that takes into account historical data access information and ensures predefined Quality of Service requirements to find a cost-efficient storage solution. Furthermore, we present a highly scalable heuristic approach that can be used with big amounts of data. The heuristic approach aims at scalability while still providing a cost-efficient storage solution that comes close to the optimal solution provided by the global optimization. As an extension to the heuristic approach we also describe a latency consideration approach. This approach incorporates latencies between the middleware and the used cloud storages into the optimization in order to find a storage solution that offers the lowest possible latencies. Therefore, customers can access their data faster which leads to a better end-user experience.

We extensively evaluate all three optimization approaches and thereby show the benefits of our designed approaches. The evaluations are presented by comparing our optimization approaches to a baseline that follows a state-of-the-art approach. We prove the correct functionality of the approaches by a detailed analysis of the results and show that we save more than 35% of total cost in comparison with the baseline.

Contents

CHAPTER 1

Introduction

"Setting goals is the first step in turning the invisible into the visible."

Tony Robbins

The use of cloud storage services to store data is nowadays a popular alternative to traditional local storage systems. In comparison to local storage systems, users of cloud-based storage solutions can benefit from a lot of different advantages. Some of the benefits that such systems bring along are better data integrity, higher availability and extended durability [6]. Another advantage is the lower cost that have to be put into administration and maintenance of the IT infrastructure, which can be particularly essential for smaller and medium-sized companies [22]. Also public institutions can benefit from the use of cloud storage services. For example, in 2009 the New York Public Library and the Biodiversity Heritage Library moved their digital content to the cloud [4].

1.1 Problem Statement

Nowadays, there exist a lot of different cloud storage providers. Amazon S3[1], Google Cloud Storage[2] and RackSpace CloudFiles[3] are just a few of the many available providers that offer cloud storages. All of these providers offer their own pricing schemes, different technologies, various geographical locations and Quality of Service (QoS) [13].

However, besides the benefits such services can offer, there are also potential drawbacks that have to be considered. One of the biggest issues when using a particular cloud

[1] https://aws.amazon.com/s3/
[2] https://cloud.google.com/storage/
[3] https://www.rackspace.com/cloud/files

storage provider is the dependence on this provider. This situation is called *vendor lock-in*, where storing a data set on only one single cloud storage provider can cause difficulties in case the provider goes out of business, is temporarily not available or increases the storage price [1, 10, 34, 47]. In this case the data has to be migrated to another cloud storage provider if the data is not already lost or inaccessible, which would be the worst case scenario.

Another problem for customers is the selection of the right cloud storage provider since a lot of different properties have to be considered. Some of the properties comprise storage technologies, pricing models and geographical locations. For instance, a customer might not be able to select a specific cloud storage provider due to internal company policies, e.g., specific data has to be stored locally or at least in the same country. Another important selection criteria is the access frequency of the data. Frequently accessed data needs to be stored on a cloud storage provider which offers low latency and little transfer cost, whereas this is not crucial for less frequently accessed data, e.g., backups or archives [16].

Going one step further by avoiding to rely on a single cloud storage provider brings up an approach which combines multiple cloud storages to distribute and store the data redundantly. This approach solves the problem of vendor lock-in. Furthermore, by incorporating a system between the data and the different cloud storage providers, i.e., a middleware, the usage of the data can be monitored, while increasing data redundancy and lowering cost. An algorithm can always choose the best-fitting, i.e., a cost-efficient, set of cloud storage providers while respecting predefined QoS constraints.

1.2 Aim of the Thesis

The aim of this thesis is to develop an optimization algorithm that dynamically optimizes the placement of data objects in the cloud. This algorithm will be integrated in a middleware that stores data objects among different cloud storage providers in a highly available, highly durable and redundant way.

The result of the data object placement algorithm, which will be integrated in the middleware, should be a cost-efficient placement solution. For achieving this requirement, the middleware optimizes the data object placement dynamically, while respecting predefined QoS attributes. Furthermore, the access frequency of the data objects is monitored in order to facilitate dynamic optimization.

For the data object placement, this work will develop and evaluate two different optimization approaches. The first one is a global optimization algorithm that always provides a cost-efficient cloud storage provider set, i.e., the optimization algorithm selects a set of cloud storage providers and distributes the data objects among the selected providers, such that the cost is as low as possible and QoS constraints are fulfilled.

The second approach is a heuristic placement algorithm for the global optimization. This heuristic approach does not necessarily always select the optimal set of cloud storage

providers, but selects the optimal set in the majority of all cases. The exact algorithm to achieve this will be evaluated during the work of this thesis.

After the conceptualization of the global optimization and the heuristic model, these two models will be integrated into the middleware and evaluated in terms of a cost analysis. This analysis is done by using a realistic set of cloud storage providers.

Moreover, one of the two placement optimizations will be extended by incorporating latencies into the algorithm. Besides cost efficiency, this extension optimizes the data placement also in a way that transfer latencies of the data objects are minimized. This extension will also be evaluated.

During the work of this thesis, the middleware CORA [49], proposed by Waibel et al. will be used as basis for the dynamic placement of data objects. CORA is described in more detail in Chapter 3.

1.3 Methodological Approach

The methodological approach for achieving the desired results is as follows:

Literature research In the first step, information of related literature is gathered in order to have sufficient background information regarding the topic of this thesis. Especially literature work in the area of cloud-based storages and multi-cloud storage systems are taken into account. Research is done using the guidelines proposed by Kitchenham and Charters [26].

Analysis of optimization models This step includes the analysis of both local and global optimization approaches. The analysis focuses on models that make use of linear and heuristic approaches.

Conceptualization of a global optimization model The conceptualization of the global optimization model is designed in a way, such that defined QoS constraints are taken into account. These constraints especially comprise availability, durability and vendor lock-in factor.

Conceptualization of a heuristic approach Based on the global optimization model, a heuristic approach is provided. This approach will consider the same QoS constraints as the global model.

Integration of the optimization models After creating the concept of the global optimization and the heuristic model, the two approaches are integrated into an existing middleware.

Evaluation of the optimization models For the evaluation of the designed and integrated optimization models, a realistic set of cloud storage providers is used.

Extension of one placement optimization One of the placement optimizations is extended in a way that latencies are also considered when selecting the provider set and distributing the data objects.

Evaluation of the extended placement optimization The extended optimization model (incl. latency consideration) is evaluated. This evaluation will use pre-measured information, i.e., the usage of all data objects is monitored and stored in a local database in order to improve the data object placement by taking into account the measured information.

1.4 Structure of the Thesis

The remainder of this thesis is structured as follows. In Chapter 2 we provide the reader with useful background information regarding the topic of this thesis. We explain general terms in the area of cloud computing and cloud-based storages as well as definitions of optimization problems and the technical specification of erasure coding.

Then we provide an overview about existing state-of-the-art approaches to cloud-based storage middlewares in Chapter 3.

In Chapter 4 we present the design and concept of the optimization algorithms that are developed to optimize the data object placement.

These developed optimization algorithms are then implemented into an existing cloud-based storage middleware. Chapter 5 will describe how the implementation is done in detail.

In Chapter 6 we will evaluate the middleware including the previously implemented optimization algorithms. The evaluation is based on a realistic set of cloud storage providers and a real-world cloud-based storage access trace. In the evaluation we will compare our optimization approaches with a baseline and discuss the results in detail.

Finally, in Chapter 7 we will conclude the thesis with a summary of the presented work and give an outlook on future work.

CHAPTER 2

Background

"The science of today is the technology of tomorrow."

Edward Teller

In this chapter we will present background information in the area of cloud computing and cloud-based storages. This serves as foundation for understanding the related technologies and concepts behind the main topic of this thesis, which is the conceptualization and implementation of an optimization algorithm for dynamic data object placement in the cloud. We will first define cloud computing and explain cloud-based storages in greater detail. Then we will discuss optimization problems. After this, we will explain the concept of erasure coding. In the last section we will define QoS and give some examples for QoS constraints.

2.1 Cloud Computing

During the last decade, more and more companies have moved away from acquiring and running their own IT infrastructure. Instead, they make use of the large number of cloud computing providers that offer their services over the Internet [43]. One of the biggest advantages of cloud computing is the ability to easily access services on demand. Cloud computing customers can buy exactly as much computing resources as they need and no longer have to purchase expensive IT infrastructure that needs a lot of time and effort for setup as well as ongoing administration [14].

For cloud users, it is easy to dynamically scale their applications running in the cloud since computing resources can be added and removed on short notice. This is a big difference compared to traditional IT infrastructure that runs inside a company. Using traditional

IT infrastructure, companies have to design the size and power of the infrastructure based on the peak load induced by their clients in order to be able to serve all incoming requests. This leads to notable over-provisioning of the infrastructure during the majority of time (i.e., when the system load is smaller than the peak load). Hence, a lot of cost are wasted and could potentially be saved. On the other hand, when the system load exceeds the maximum load the system can handle, the system will fail to serve all requests and some users will not be served by the system. Therefore, these users do not generate any revenue and maybe they will never return to use the system again. With cloud computing, companies can dynamically adjust their infrastructure according to the system load. Scaling up/down can be done automatically when it is detected that more/less resources are needed [6].

Most cloud computing providers, such as Amazon[1] and Microsoft[2], offer a pay-per-use pricing model. Users of their cloud computing services are only charged for the amount of resources they actually use. This especially lowers the barrier to entry for most users as they do not need to acquire expensive IT infrastructure in order to be able to test or run their own applications. Instead, users can rent services from cloud computing providers and only have to pay a small subscription fee for the time of the usage. When the cloud computing services are no longer needed, users can stop using them without the need of spending additional budget. Once the usage of particular resources stop, the billing for the usage of these resources stops as well [20].

When it comes to finding a universal definition, it can be said that there exist a lot of different meanings of the term *cloud computing*. However, the most suitable definition, in our opinion, is the one given by the National Institute of Standards and Technology (NIST) [35]:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The reason for the existence of a lot of different definitions for the term cloud computing is the fact that cloud computing itself is not a new technology but rather a paradigm that makes use of multiple tools and technologies. For instance, cloud computing utilizes virtualization and utility-based pricing in order to offer a flexible service model for cloud computing users [53].

As the main emphasis of this thesis lies on cloud-based storages, in the following we want to focus on cloud-based storage services rather than on cloud computing in general.

[1] `https://aws.amazon.com/pricing/`
[2] `https://azure.microsoft.com/pricing/`

2.2 Cloud-based Storages

Besides computing services, storage services are one of the most used services by cloud computing users [17]. Cloud-based storages are used to store data in the cloud (i.e., on cloud infrastructure that is maintained by a cloud computing provider). The provider that is responsible for storing the data safely and making it available and accessible to the client, which can be an individual or a company, is called *cloud storage provider*. The according service model is often referred to as Storage as a Service (StaaS). With this service model, the cloud storage provider rents storage space to an individual or another company. The infrastructure, where the actual data is stored, can be distributed around the whole globe [37].

Typical use cases for using cloud-based storages are data backup, data archival or simply using the cloud storage as a replacement for local storage (e.g., if there is not enough storage capacity available locally). Usually, data in the cloud may be accessed via an Application Programming Interface (API) or via a web-based management platform [52].

2.2.1 Benefits

Some of the most important benefits of cloud-based storages in comparison to traditional (i.e., physical) storage systems are listed in the following [22, 28]:

Cost reduction Organizations only have to pay for the storage they actually use. When a company uses physical storage, it has to acquire high-capacity storage infrastructure with enough storage capacity to store all potential data. In contrast, since cloud storage providers use a pay-per-use pricing model, companies only pay for the storage they actually use.

Reduced energy consumption Companies can reduce their energy consumption by up to 70% when replacing their in-house storages with cloud-based storages, which also leads to cost savings in the end.

Convenience Companies do not need to install and run expensive infrastructure in their own offices. By using cloud-based storages, companies can transfer the maintenance responsibility to the cloud storage provider. This way, the company can focus on its key business and does not need to waste time on tasks such as purchasing additional infrastructure or fixing issues with the storage system. Instead, the storage infrastructure is managed by dedicated experts.

Accessibility By using cloud-based storages, data is made accessible over the Internet. This way, also employees that work outside the company can easily access the data from potentially anywhere. Moreover, sharing information with business partners becomes easier. Setting up File Transfer Protocol (FTP) access becomes obsolete, information can be shared with partners by simply providing a link to the data.

Reliability If a company uses physical storage systems, system failures may be undetected for a long time (e.g., when a failure happens during the night). Usually, cloud storage providers agree to provide a minimum availability rate per month. Thus, the provider has to monitor and ensure this objective permanently, leading to improved availability and reliability.

2.2.2 Drawbacks

In contrast to the before mentioned benefits, cloud-based storages also bring along some drawbacks [6]:

Security With the use of cloud storages, data is stored in locations outside of the data owner's responsibility. This implies that the cloud storage provider has to ensure the according level of security such that it is not possible for unauthorized third parties to access data owned by others. Since most of the cloud infrastructure is shared among multiple cloud users, the cloud storage provider has to guarantee that unauthorized users are not able to access foreign private data. Usually, this is realized by virtualization of the infrastructure.

Privacy Since the cloud storage provider operates the infrastructure where data is stored on, it is inevitable that the provider can theoretically access this data. This fact can potentially hold back customers from storing their data on a cloud storage. However, due to the terms of agreement between the cloud storage provider and the cloud user, the provider is normally not allowed to access the stored data. Still, a solution to increase the level of privacy is to encrypt data before storing it on a cloud storage.

Vendor Lock-in The situation when data is locked in to one specific cloud storage provider should be avoided. Due to the fact that most providers offer their own proprietary APIs to store and access data, it is hard to migrate data from one provider to another. Furthermore, if a provider experiences an outage, customers are not able to access their data until the outage is fixed.

Due to the mentioned drawbacks of using cloud storages, in this thesis we design and implement a data object placement algorithm that uses multiple cloud storages instead of only one storage. By using multiple cloud storages, it can be ensured that security and privacy requirements are met and that vendor lock-in is avoided. For achieving this, we use erasure coding, which will be described in Section 2.4.

2.2.3 Pricing

There exist a lot of different cloud storage providers and each provider offers its own pricing scheme. Standard storage services, such as Amazon S3, Microsoft Azure Storage and RackSpace CloudFiles offer a very similar pricing scheme. In contrast, there also exist

long-term storage services such as Amazon Glacier, which are specifically designed to store rarely accessed data. On one hand, long-term storage services have higher transfer cost, but on the other hand, they offer a reduced storage cost that makes them eminently suitable for rarely accessed data.

Despite the different pricing models, most of the schemes are based on a similar concept. The total price is based on the amount of used storage, the transferred data (in most cases only outgoing transfer is billed) and the number of read and write requests. Most of the providers do not charge anything for incoming transfer or deleting data, they only charge for outgoing transfer. Furthermore, most providers introduce a *Block Rate Pricing model* [37]. With this model, additional storage and bandwidth are cheaper, the more capacities are leased. That is, the more data is stored and transferred, the lower is the price per GB. This model can be particularly important for companies that store huge amounts of data on cloud storage providers as they can save a lot of money making use of this pricing model.

Especially large cloud storage providers offer the possibility to store data in different geographical locations. When data needs to be migrated from one location to another, the majority of the providers offer a reduced migration price.

As already mentioned before, an alternative to standard storage offerings are long-term storage services. These services are designed specially for data that is rarely accessed or not accessed at all. Thus, the storage price is lower, but the access price is higher than for standard storage services. Furthermore, long-term storage services often have a Billing Time Unit (BTU) (i.e., a minimum storage duration). This means that when a file is stored for a time shorter than the BTU, the storage provider still charges the full duration of the BTU. Long-term storage systems may also have a Billing Storage Unit (BSU) (i.e., a minimum object size). For data objects smaller than the BSU, the charging is done for the full BSU.

Numerous cloud storage providers also offer different storage classes for their services. As an example pricing model, in the following we will describe the Amazon S3 pricing model in detail. For instance, besides its Standard Storage, Amazon also offers Reduced Redundancy Storage, Standard - Infrequent Access (IA) Storage and the previously mentioned Glacier Storage. Since Standard - IA Storage has a lower storage price, but also reduced availability, it makes the service suitable for long-term storage, backups and as a data store for disaster recovery [49].

Table 2.1 shows the recently updated pricing model [7] of the Amazon S3 Standard Storage compared among different regions. From the table, one can see the Block Rate Pricing model that is applied depending on the amount of used storage space. The more storage space is leased, the cheaper is the price per additional storage space.

The bandwidth price of outgoing data transfer is shown in Table 2.2. Similar to the storage price, the outgoing bandwidth price also follows the Block Rate Pricing model, that makes additional data transferred from S3 to other locations cheaper.

Table 2.1: Amazon S3 Storage Price for Standard Storage

Usage	Price in $/GB/month			
	N. Virginia	Frankfurt	Tokyo	Sao Paulo
First 50 TB	0.0230	0.0245	0.0250	0.0405
Next 450 TB	0.0220	0.0235	0.0240	0.0390
Over 500 TB	0.0210	0.0225	0.0230	0.0370

Table 2.2: Amazon S3 Outgoing Bandwidth Price for Standard Storage

Usage	Price in $/GB/month			
	N. Virginia	Frankfurt	Tokyo	Sao Paulo
First 1 GB	0.000	0.000	0.000	0.000
Up to 10 TB	0.090	0.090	0.140	0.250
Next 40 TB	0.085	0.085	0.135	0.230
Next 100 TB	0.070	0.070	0.130	0.210
Next 350 TB	0.050	0.050	0.120	0.190
Next 524 TB	On demand	On demand	On demand	On demand
Next 4 PB	On demand	On demand	On demand	On demand
Greater than 5 PB	On demand	On demand	On demand	On demand

It is also shown in both tables that prices differ quite significantly among different geographical regions. For instance, outgoing bandwidth price in Sao Paulo is about three times as costly as in Northern Virginia or Frankfurt.

Amazon, as well as other cloud storage providers, offers a reduced migration price if data is moved from one region to another. For instance, for migrating data from Northern Virginia to Frankfurt, Amazon charges $0.020 per GB. Depending on the location of the user, the region of the data storage can get quite important. Obviously, latency will be lower if data is located closer to the user [41]. If data is stored in a region far away from the user, then data operations will take longer and bandwidth may be lower. Later in this thesis we will consider a scenario in which it makes sense to migrate data at the cost of increased storage or bandwidth price from one region to another in order to achieve reduced latency (see Section 4.5).

2.3 Optimization Problems

In order to understand the central goal of this thesis, it is important to clarify the term *optimization problem*. An optimization problem is the problem of finding the best possible (i.e., optimal) solution. Usually there exist multiple solutions for a given problem, but the goal of an optimization problem is to find the best solution of the set of all feasible

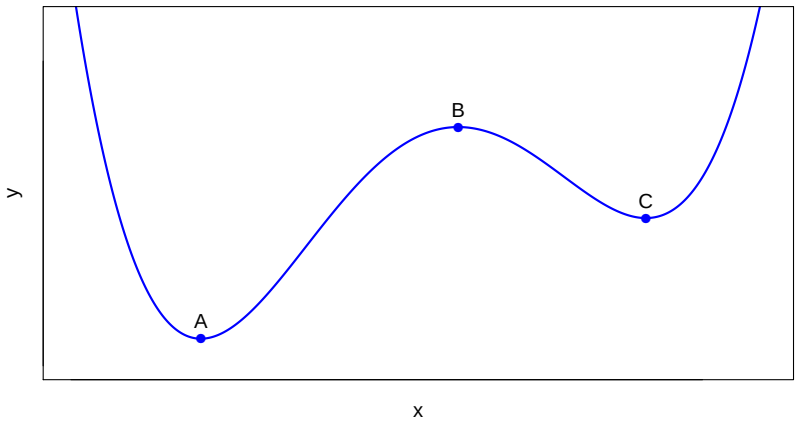


Figure 2.1: Local and Global Extrema of a Function

(i.e., possible) solutions.

2.3.1 Local vs. Global Optimization

In Figure 2.1 the difference between local and global extrema is shown. The displayed function is a polynomial of fourth degree. Point A is the *global* minimum of the function since the corresponding function value is smaller than every other function value. Point B is a *local* maximum since it is the greatest function value within a given neighbor range around the point. B is not a global maximum since it can be seen that there exist greater function values left to A and right to C. Point C is a *local* minimum since it is the smallest function value within a given neighbor range around the point, but C is not a *global* minimum since the function value of A is smaller.

Thus, we have found the following extrema:

- A: global minimum
- B: local maximum
- C: local minimum

2.3.2 Examples of Optimization Problems

In the following we will discuss two well-known optimization problems, the Traveling Salesperson Problem and the Knapsack Problem.

Traveling Salesperson Problem

A well-known example for an optimization problem is the Traveling Salesperson Problem (TSP). Given a set of n cities and the distance between every pair of cities, the traveling

salesperson has to plan a round trip that starts and ends in the same city. The goal is to find the cheapest (i.e., in terms of shortest distance) trip that visits each city exactly once. Since the traveling salesperson has to find the trip with shortest distance, the problem can be formulated as a minimization problem [23, 24].

Speaking in terms of functions, TSP can be seen as the problem of finding the global minimum of a function. Analogous to Figure 2.1, the problem would be to find the value of Point A, since we already know that A is the global minimum of the function. Another feasible solution would be to find the value of C. Since C is just a local minimum, but not a global minimum, C is just the solution for a local optimization (i.e., minimization) problem, but not for the global optimization problem.

Knapsack Problem

Another popular optimization problem is the so-called Knapsack Problem (KP). The problem deals with the fact that a hiker needs to pack a knapsack for a mountain tour. She has a number of items n available and wants to pack the most valuable items into the knapsack. Each item $j \in \{1, \ldots, n\}$ has a value v_j and a weight w_j. The goal of the hiker is to maximize the total value of all packed items, while respecting the maximum capacity of the knapsack W. Thus, the problem can be formulated as follows:

$$\text{maximize} \sum_{j=1}^{n} v_j x_j \tag{2.1}$$

$$\text{subject to} \sum_{j=1}^{n} w_j x_j \leq W \tag{2.2}$$

$$x_j \in \{0, 1\}\,,\ j = 1, \ldots, n \tag{2.3}$$

where $x_j = 1$ if item j is packed into the knapsack, and $x_j = 0$ otherwise.

The knapsack problem is a typical example of a maximization problem, as the goal of the problem is to find the solution with maximum value. The basic knapsack problem is constructed with one constraint (i.e., the maximum capacity of the knapsack W), but there exist a lot of variations of the knapsack problem. The problem that we are going to make use of later in this thesis is the Multidimensional Knapsack Problem (MKP). The multidimensional knapsack problem extends the basic knapsack problem by introducing multiple constraints instead of one single constraint. We assume that the hiker has a total number n of items and a total number m of knapsacks available. Out of all items, she has to pack those items with maximum value into the knapsacks, while the packed items can not exceed the maximum capacity W_i of each knapsack. It can be formulated

as follows:

$$\text{maximize} \sum_{j=1}^{n} v_j x_j \tag{2.4}$$

$$\text{subject to} \sum_{j=1}^{n} w_{ij} x_j \leq W_i,\ i = 1, \ldots, m \tag{2.5}$$

$$x_j \in \{0, 1\},\ j = 1, \ldots, n \tag{2.6}$$

where $x_j = 1$ if item j is packed into knapsack i, and $x_j = 0$ otherwise [25].

The MKP is known to be NP-complete [19]. Therefore, the solution to this problem is far from trivial. There exist multiple approaches of solving the MKP. Exact approaches have the problem that they are computationally very complex and usually require long runtime. Thus, commercial software (including CPLEX[3] and others) has been developed to provide an easier-to-use interface for solving optimization problems. In order to be able to provide the software with an optimization problem, one has to formulate the according problem as an integer programming model. In integer programming models, variables are restricted to be integers. A special version of integer programming is Integer Linear Programming (ILP), where the objective function and the constraints are linear. In this thesis we will use Mixed Integer Linear Programming (MILP) to solve optimization problems. In contrast to classical integer programming, MILP does not restrict all variables to be integers, but also allows some variables to be non-integers.

Besides exact methods, there also exist heuristics to solve the MKP. Heuristics aim at providing a near-optimal solution to the optimization problem by using less cost (i.e., finding a solution within less time and consuming less memory). The big advantage of heuristics are that they usually provide feasible solutions that are within an acceptable deviation of the optimal solution while computing a solution takes less time compared to exact methods. Thus, heuristics are more useful in practice when it comes to resource effectiveness and performance [18].

In Chapter 4 we will go into more detail about the exact optimization problem that will be solved during the work of this thesis.

2.4 Erasure Coding

The requirements for today's IT solutions range from high availability to good performance and good reliability. Highly reliable systems have to be designed in a way to be able to gracefully react to system failures. This is typically done by incorporating some sort of redundancy into the system. The redundancy of data objects is one of the central topics of this thesis. In this section we introduce the concept of erasure coding, which provides the ability to increase system reliability while, at the same time, decreasing the amount of storage space needed.

[3] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

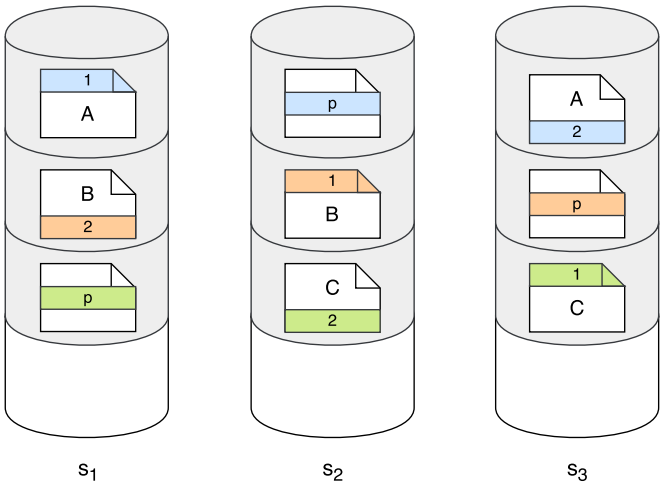


Figure 2.2: Erasure Coding by the Example of a $(2, 3)$ Coded System

A straight-forward approach to increasing data reliability of a system is full replication. With full replication, data is replicated multiple times and the same replicas are stored on different storages. For instance, assume we want to enhance the system reliability by a factor of 3. Then we have to store the same replica (i.e., a copy of the whole data) on three different storages. Thus, reliability of the system is enhanced by a factor of 3 since all three storages have to break at the same time in order to break the whole system. However, the downside of this method is that it also occupies three times as much storage space as a system without redundancy [54].

2.4.1 Definition

A more economical solution to full replication is erasure coding [42, 44, 46, 48]. An erasure code is defined by a tuple (m, n). Each data object is split into m parts and is then encoded into n data object chunks, where any subset of size m $(m < n)$ is sufficient to reconstruct a copy of the whole data. Thus, the erasure coded system is capable of tolerating up to $n - m$ simultaneous storage failures. This benefit comes at the price of additional storage need. If we assume that the storage space without erasure coding is m, then the additional storage space induced by erasure coding is $k = n - m$. Hence, the total storage space needed with erasure coding is $n = m + k$. The encoding rate of an (m, n) erasure code is defined as $r = \frac{m}{n} < 1$. An erasure code with rate r leads to an increase in storage space of factor $\frac{1}{r}$.

For example, with a $(2, 3)$ erasure code, a data object is split into $n = 3$ data object chunks and distributed among three storages. If any of the storages fails, the remaining two storages are sufficient to fully reconstruct the original data.

Figure 2.2 shows the concept of erasure coding by the example of a $(2, 3)$ coded system.

Each data object A, B and C is split into three data object chunks. Two of these three chunks, labeled as 1 and 2, are data chunks and the remaining chunk, labeled as p, is a parity chunk. These three chunks are distributed among three different cloud storages s_1, s_2 and s_3. The chunks are placed in a way such that the outage of any of these three storages can be tolerated, i.e., each of the three chunks of one data object is placed on a different storage.

As erasure coding allows arbitrary combinations of m and n, we can, for instance, construct full replication with a replication factor of 3 by setting $m = 1$ and $n = 3$, thus yielding in a $(1, 3)$ coded system.

Another popular storage technology is called Redundant Array of Independent Disks (RAID) [40]. RAID combines multiple physical disk drives to a single logical unit to improve data redundancy and performance. Since erasure coding is a super set of RAID, we can also describe every RAID system by erasure codes. For example, RAID-5 can be described as a $(4, 5)$ erasure code [39, 50].

2.4.2 Benefits

Later in this thesis we will use erasure coding to encode data objects and distribute the resulting data object chunks among multiple cloud storage providers. Here we want to briefly mention the reason for this decision and outline the benefits of erasure coding over single storages and full replication.

Obviously, single storages cannot provide the same reliability as a redundant storage system. With one single storage, there exists a Single Point of Failure (SPoF). The problem of a SPoF is that when this component fails, the whole systems becomes unavailable.

The basic approach to avoid a SPoF is the introduction of redundancy by replicating the original data (i.e., full replication) to a second storage. This eliminates the SPoF and provides enhanced reliability since a system outage only occurs when all storages fail at the same time. However, full replication brings along the disadvantage of occupying considerably more storage space. By introducing one replica additionally to the original storage (i.e., two storages in total), the whole system also needs twice the amount of storage space (i.e., 200%) compared to one single storage.

In contrast, erasure coding only needs a fraction of additional storage space. For example, the use of a $(2, 3)$ erasure code means that the original data is divided into three parts and stored among three different storages. Obviously, each storage now only needs a third of the original storage space. As already described previously, an erasure coded system can tolerate up to $n - m$ failures. In this case, the system could tolerate the failure of any of the storages, which yields in the same property as a full replication system with 2 replicas. However, what sets erasure coding apart from full replication, is the fact that an erasure coded system needs less additional storage space in comparison to full replication. With the before mentioned examples, a $(2, 3)$ coded system needs

$\frac{1}{r} = \frac{n}{m} = \frac{3}{2} = 150\%$, while full replication with 2 replicas needs 200% of storage space (in comparison to a single storage).

One issue of erasure coding that has to be considered is performance. Since each data object is split into m parts and encoded into n data object chunks, data encoding can become computationally expensive. Therefore, erasure coding might not be the best fit for real-time systems or applications where performance is essential for ensuring high usability. Still, for our purpose, erasure coding fits perfectly since we mainly focus on placement optimization of data objects instead of achieving best performance [2].

2.5 Quality of Service

When designing a middleware that uses different cloud storages to store data, it has to be ensured that a variety of system requirements are fulfilled. This can be achieved by formulating QoS constraints. These defined constraints have to be met by the system. The following QoS constraints are taken into account while designing the optimization algorithms during the work of this thesis:

Data durability In most cases, loss of previously stored data is fatal and will not be tolerated by users. Data durability defines the probability that stored data does not get permanently lost, e.g., a data durability of 99.99999% per year means that with a probability of 99.99999% data will be stored permanently at least for one year. In other words, a data loss will occur with a probability of $1 - 0.9999999 = 0.0000001 = 0.00001\%$ [38].

Data availability Data is considered available, if users can successfully access the data. Normally, high availability is a key requirement by users of a system. It is important to outline that durability does not imply availability (i.e., data can be successfully stored and subsequently accessible, but not accessible at a given point in time). On the other hand, availability requires durability (i.e., in order to be accessible, data has to be stored in advance to retrieving it). Data availability is defined as the probability that data is accessible, e.g., a data availability of 99.99% per year means that with a probability of 99.99% data can be accessed at any given point in time within this year. In other words, data will not be accessible on $1 - 0.9999 = 0.0001 = 0.01\%$ of the year [38].

Vendor lock-in factor If data is stored on only one cloud storage provider and cannot be moved to another provider, we consider the data as locked in to a provider. The vendor lock-in factor *lockin* is defined as follows: $lockin = \frac{1}{N}$, where N is the number of cloud storage providers where a particular data object is stored. The more cloud storage providers are used for storing one particular data object, the smaller (and therefore better) is the vendor lock-in factor. At most, the vendor lock-in factor can be 1. This is the case if only one single provider is used for storing a data object, what should apparently be avoided [54].

Latency In real-world systems there is naturally always a delay between sending data and receiving data (e.g., due to physical laws). If we consider a request-response interaction between two parties A (sender) and B (receiver), we define latency as the end-to-end Round-Trip Time (RTT). The RTT is the elapsed time starting from sending a request from A to B until the response of B arrives back at A. Latency is specified as a time period, e.g., a latency of 100ms means that sending a request from A to B, processing the request at B and returning the response from B to A altogether takes 100ms [41].

Usually, customers of cloud storages define their distinct Service Level Objectives (SLOs). For every kind of QoS constraint, they define the minimum value that the service has to guarantee. The central goal of this thesis is to minimize the overall storage cost, while meeting all customer-defined SLOs. Specifically, cost minimization among multiple cloud storage providers is the main objective of the optimization problem, while the customer-defined SLOs function as constraints of the model [3, 36, 49].

CHAPTER 3

Related Work

"The power to question is the basis of all human progress."

Indira Gandhi

In this chapter we want to analyze related work in the area of cloud-based storage systems. We focus on approaches that are relevant for this thesis and present closely related work. The focus of the analysis lies on cloud-based storage solutions that store data in a redundant manner. First we will discuss approaches that aim at cost optimization. Then we will explain one solution that also considers latency in the optimization approach. Finally, we will investigate solutions that do not aim the optimization at minimizing cost.

3.1 Cost Optimization

In the following we will discuss related work with focus on cost optimization.

CORA

As already mentioned in Section 1.2, CORA [49] will be used as basis for dynamic data object placement on multiple cloud storages. CORA is the most recent approach among all related work and aims at minimizing the overall cost of redundant data storage in the cloud. For storing the data in the cloud, CORA uses a set of cloud storages. These cloud storages may be public or private. CORA addresses the problem of cost-efficient data redundancy in the cloud by introducing an optimization problem that aims at optimizing the placement of data objects on different cloud storages. The optimization ensures that user-defined QoS constraints (i.e., data durability, data availability and vendor lock-in factor) are fulfilled. Moreover, CORA also considers the data access pattern of different

data objects in order to improve the data object placement. For achieving this, CORA continuously monitors the usage of the data objects and dynamically rearranges the placement if a better (i.e., a new cost-efficient) solution is found. CORA uses erasure coding as a redundancy mechanism to distributively store data among several cloud storages. What sets CORA apart from other redundant cloud storage approaches, is the ability to store rarely accessed data on long-term storages instead of placing them on standard storages. This approach makes CORA save money since long-term storages usually offer a lower storage cost compared to standard storages. Furthermore, CORA can work with the Block Rate Pricing model used by different cloud storage providers, which can help to reduce cost for saving big amounts of stored data. Therefore, CORA perfectly fulfills the requirements of this thesis. For this reason, CORA will be used as the middleware in this work. Our optimizations will be integrated into the middleware and evaluated as described previously.

Scalia

The work which comes closest to CORA is Scalia [39]. Scalia is a redundant cloud storage system that also aims at minimizing the data storage cost. The placement algorithm proposed by Papaioannou et al. is based on real-time data access patterns that are regularly stored in a persistent database. Rules that are defined by the data owner are also taken into consideration when optimizing the placement. Scalia computes the cheapest cloud storage provider set among the whole range of possible alternatives. Similar to CORA, Scalia also uses erasure coding to split data objects into smaller redundant data object chunks and to distribute these data object chunks to multiple cloud storages. Scalia is structured in three layers. One of the three layers is a caching layer that can greatly improve read performance by caching previously read data. At the same time, it can reduce cost by not directly propagating each request to the cloud storage provider. Instead, cached data objects can be delivered by the middleware and do not always incur a read request at the cloud storage provider. For the placement optimization, the authors formulate an optimization problem that resembles the Multidimensional Knapsack Problem. Instead of MILP, which is used by CORA to find the set of optimal cloud storage providers, the authors use a heuristic approach for this in Scalia. The heuristic approach takes into consideration access patterns of the data objects by saving historical data to a persistent database. This approach is particularly important as it is also in the scope of this thesis, as described in Section 1.3. However, what sets CORA apart from Scalia is the fact that CORA distinguishes between standard storage and long-term storage solutions in order to particularly save cost for data objects that are accessed very rarely. Moreover, Scalia does not make use of the Block Rate Pricing model but rather uses a simplified pricing model that does not resemble real-world pricing schemes.

RACS

Abu-Libdeh et al. propose a cloud-based storage middleware called RACS that makes use of erasure coding [1]. The authors were first in the field of avoiding vendor lock-in by introducing a redundant middleware that makes use of erasure coding for distributing data objects among multiple cloud storage providers, while considering basic cost models of the providers. They argument that RAID-like systems have been successfully running with physical storages in the past in order to improve redundancy. The authors use the already established approach of RAID and transfer it to cloud-based storages with the usage of erasure coding. In their system design, Abu-Libdeh et al. argue that it is sufficient to add enough redundancy to be able to tolerate one provider outage at a time due to the high up-time percentages of the cloud storage providers. In comparison to CORA and Scalia, RACS does not monitor the usage pattern of data objects. Thus it cannot use the access information to improve the placement of data objects. Moreover, compared to CORA, RACS (as well as Scalia) does not differ between standard storages and long-term storages.

CHARM

With CHARM [54], Zhang et al. also introduce a multi-cloud storage middleware. Similar to the before mentioned approaches, CHARM also aims at avoiding vendor lock-in and thereby focuses on cost efficiency. CHARM considers different redundancy mechanisms such as replication and erasure coding. The system uses the access history of a data object to determine the best fitting storage mechanism for this particular object. Depending on the read frequency and the size of a data object, CHARM selects the storage mode (i.e., replication or erasure coding) that fits best for this particular data object. That is, based on read frequency and data object size, the system selects the cheapest storage mode. For the evaluation of the system, the authors provide a comparison between redundancy, erasure coding and CHARM for different use cases. Comparable to Scalia, CHARM also uses a simplified pricing model that makes no difference between standard and long-term storage solutions and does not consider the Block Rate Pricing model. This leads to the same limitations as previously discussed for Scalia.

DepSky

Another cloud storage system that stores data on multiple cloud storage providers is DepSky, as proposed by Bessani et al. [11]. This system, similar to previously presented approaches, aims at improving the availability and reliability of data storage in the cloud by using replication mechanisms. The authors focus on integrity and confidentiality of the data and address the topic of storing critical data in the cloud, while putting an emphasis on security and integrity. In their work, they also use erasure coding for distributing the data among multiple cloud storages in order to overcome the problem of vendor lock-in. The authors also aim at minimizing the cost incurred by overall data storage, but they do not consider Block Rate Pricing models of the various cloud storage

providers. Instead, the authors assume that the storage cost is directly proportional to the amount of stored data, which does not resemble the cost models in reality.

CAROM

Combining both replication and erasure coding in one system, named CAROM, is done by Ma et al. [30]. The authors propose a system that includes a cache to store recently accessed data. The cache is used to reduce bandwidth cost incurred by frequent reconstruction of data objects out of the according data object chunks. This is done in a similar way as in Scalia. Ma et al. state that the integrated cache reduces access latency. The authors of CAROM introduce a heuristic that dynamically adapts the cache size depending on the access patterns of the stored data objects. Besides lower access latency, another big advantage of integrating a cache into a cloud-based storage middleware is cost reduction due to bandwidth savings. If the cache is designed appropriately, the system does not have to propagate every request directly to the cloud storage providers where chunks of the requested data object are stored. Instead, the cache can be used to deliver data directly to the client, as well as storing requested data objects for a limited amount of time. For the evaluation of the system, the authors compare CAROM against full replication and erasure coding. They state that their system outperforms both alternatives in terms of total cost. However, performance of their system strongly depends on the cache hit rate, i.e., the percentage of data accesses that can be served by the cache and therefore result in a cache hit.

3.2 Latency Consideration

In this section we discuss one optimization approach that aims at cost minimization, as the work in the previous section. Additionally, this approach also considers latency.

SPANStore

In the work of Wu et al., the authors present a key-value storage system called SPANStore [51]. The system aims at lowering storage cost by distributing replicas of data objects among various geographically distributed cloud storages. Similar to CORA, the optimization problem proposed by the authors is solved using MILP. What sets this work apart from others in the related area is the fact that the authors incorporate latency constraints into the optimization problem. In this work, the solution to the placement problem is realized as a trade-off between availability, storage price and latency. However, compared to other approaches, SPANStore does not use erasure coding for distributing data objects among multiple cloud storage providers. Instead, the authors use full replication what greatly increases the needed storage space. Furthermore, SPANStore does not take into account Block Rate Pricing models of different cloud storage providers.

3.3 Further Related Work

In this section we summarize and explain optimization approaches that do not aim at cost minimization.

HAIL

In their work called HAIL [12], Bowers et al. propose a distributed cryptographic system that is able to prove that a stored file is completely intact and can, therefore, be retrieved by a client. The described approach extends the basic functionality of RAID to protect the system against mobile adversaries. In contrast to other approaches, the authors put an emphasis on integrity and security. They use cryptographic mechanisms to verify that a particular file is retrievable, meaning that the file can be delivered to the client without any loss or corruption. In a similar way as other approaches, HAIL uses erasure coding to distributively store data among multiple cloud storage providers. For being able to verify the availability and integrity of a file, the system checks the corresponding file parts at multiple distributed servers. If it detects any kind of corruption at a given server, it can force the remaining servers to recover the corrupted file. In comparison to the previously mentioned approaches, HAIL focuses on availability and security, but neglects cost totally. This means that it cannot improve the data object placement depending on the incurred cost which leads to the fact that it cannot provide an optimal solution regarding cost efficiency. Another disadvantage of HAIL is the inability to update previously stored data. It is only capable of writing data objects once, which strongly limits the system's usability in practice.

MetaStorage

MetaStorage, a system proposed by Bermbach et al., uses a distributed hash-table to distribute data objects among different cloud storage providers [9]. The authors use full replication to redundantly store data among multiple providers. This way, MetaStorage can not be as economical as the other systems since the whole data is fully replicated multiple times among different cloud storage providers. Nevertheless, this approach still provides a high level of availability and aims at bypassing vendor lock-in by using redundancy with multiple cloud storage providers. Like HAIL, MetaStorage does not take into account different pricing models of cloud storage providers. Various cost components (e.g., storage, bandwidth and transfer cost) are completely neglected. Therefore, the system cannot provide a cost-efficient solution to the placement problem of data objects. One more drawback of MetaStorage (and HAIL) is the fact that both systems do not take data access patterns into account. Without cost consideration, data access patterns can not be used to improve the data object placement in order to get a cost-efficient placement solution.

Table 3.1: Comparison of Related Work

✓ Feature is fully covered
○ Feature is partially covered
- Feature is not covered

Feature	[49]	[39]	[1]	[54]	[11]	[30]	[51]	[12]	[9]	[15]	[32]	Our work
Cost optimization	✓	✓	✓	✓	✓	✓	✓	-	-	○	○	✓
Block Rate Pricing	✓	-	-	-	-	-	-	-	-	-	-	✓
Data access patterns	✓	✓	-	✓	-	-	✓	-	-	-	-	✓
Long-term storages	✓	-	-	-	-	-	-	-	-	-	-	✓
Erasure coding	✓	✓	✓	✓	✓	✓	-	✓	-	-	-	✓
Latency	-	-	-	-	-	-	✓	-	-	-	-	✓

Probability-Based Cloud Storage Providers Selection Algorithms with Maximum Availability

Chang et al. formulate an optimization problem that is similar to the Knapsack Problem [15]. The optimization problem aims at maximizing data availability by placing data objects on different cloud storage providers. In their work the authors present a mathematical solution to the problem. However, the solution does not take into account important cost factors such as transfer and migration cost. Moreover, the authors do not include long-term storage solutions in their optimization.

Brokering Algorithms for Optimizing the Availability and Cost of Cloud Storage Services

Similar to the prior presented approach, Mansouri et al. also focus on data availability by introducing a mathematical solution to the optimization problem [32]. They aim at minimizing storage cost while ensuring a defined percentage of data availability. Mansouri et al., as well as Chang et al., do not consider transfer cost, data migration cost and long-term storages in their optimization models. This is an essential weakness of these solutions in comparison to more sophisticated approaches such as CORA or Scalia. By doing this, they ignore important components of real-world pricing models. Therefore, they can not provide a realistic cost optimization model.

3.4 Feature Comparison

After presenting different approaches to cloud-based storage solutions in the previous sections, we now compare the presented approaches against each other.

An overview of related work is shown in Table 3.1. It can be observed from the table that most of the introduced approaches aim at cost optimization. For achieving this, most approaches use erasure coding as a redundancy mechanism. Only some of the presented

work consider data access patterns in their optimization models. As already outlined earlier, CORA is the only work that includes Block Rate Pricing models of the different cloud storage providers and includes long-term storages in the optimization model as an alternative to standard storages. What is hardly taken into account in the presented work is latency. Only the authors of SPANStore consider latency in their optimization model.

In contrast to the analyzed related work, our work will cover all features that are shown in the table.

3.5 Research Challenges

Apart from CORA, none of the before mentioned approaches provide a cloud-based storage solution that monitors the usage of data objects, while also taking into consideration long-term storages to improve the data object placement. Moreover, only CORA and SPANStore use MILP as an approach for solving the optimization problem and none of the mentioned approaches, except for SPANStore that only covers the topic partially, take latency into consideration.

In summary, this work will provide two optimization algorithms that solve the problem of cost-efficient data object placement in the cloud. Both algorithms take into consideration Block Rate Pricing models of the different cloud storage providers, data access patterns of the stored data and long-term storages in addition to standard storages for the data object placement. The aim of the algorithms is to minimize overall storage cost. For achieving this, we will use MILP as an approach to solve the optimization problem in the first algorithm. In the second algorithm we will implement a heuristic solution. As redundancy mechanism we will use erasure coding. Furthermore, we will also consider latency in the optimization models.

CHAPTER 4

Design

"Design is a funny word. Some people think design means how it looks. But of course, if you dig deeper, it's really how it works."

Steve Jobs

The goal of this chapter is to present the system design and the underlying technical models. For achieving this, we will first describe the whole system from an architectural point of view. Then we will introduce the functionality of the used cloud-based storage middleware. After that, we will introduce the system model, where we will define and describe the needed terms and variables to subsequently formulate the optimization problems.

4.1 Architectural Overview

In this section we give an architectural overview of the system structure. First we explain the overall system architecture and then we discuss the functionality of the middleware in more detail.

4.1.1 General Architecture

As already described previously, the goal of this thesis is to design and implement an optimization algorithm that optimizes the placement of data objects in the cloud. Figure 4.1 shows the overall system architecture consisting of three major parts. The first part is the user which is shown at the bottom of the graphic. The user of the system wants to store data objects (i.e., files) on cloud storages. For this, the user needs

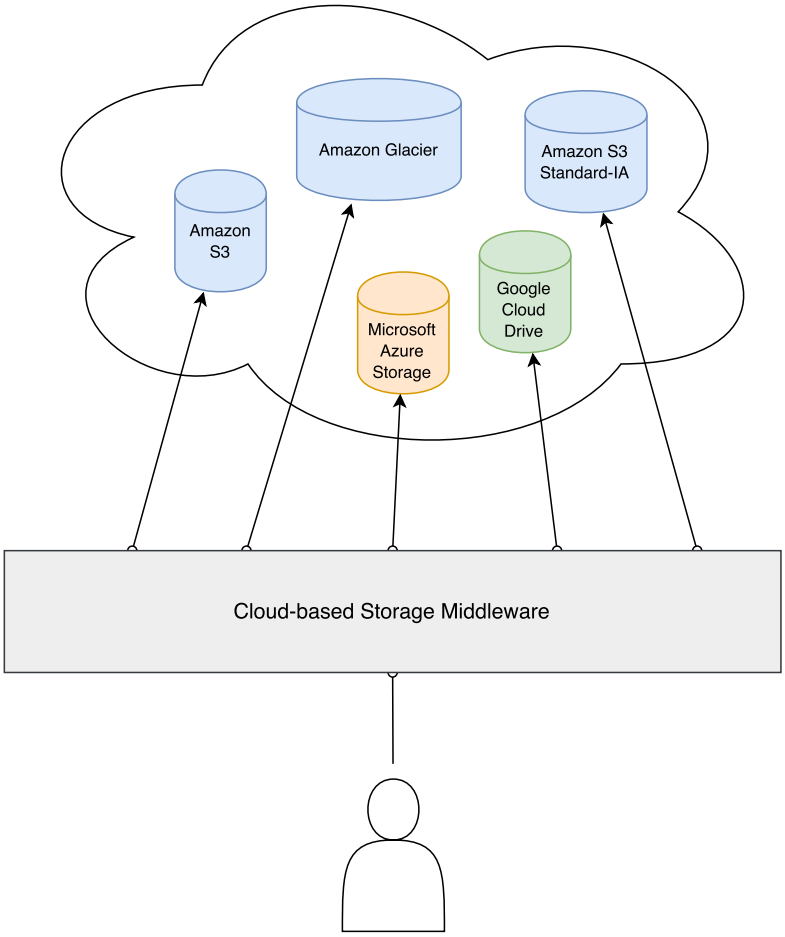


Figure 4.1: Architectural Overview of the Overall System

at least one cloud storage provider to store the files. Due to the fact that we want to prevent vendor lock-in, we use multiple cloud storage providers instead of only one single provider. Some of the various storages are shown inside the cloud at the top of the graphic. In order to distribute the data objects to multiple cloud storages, we need a component that acts as connector between the user and the different cloud storages, i.e., a middleware that requires the user to upload the data objects only once, while the middleware distributes the data objects among multiple cloud storages. The middleware (shown in the center of the graphic) marks the third part of the system. The optimization algorithm that dynamically optimizes the placement of data objects uploaded by the user will be integrated into the middleware. The result of this optimization should be a cost-efficient placement solution.

As we previously argued in Section 3.1, CORA fits the requirements of the cloud-based

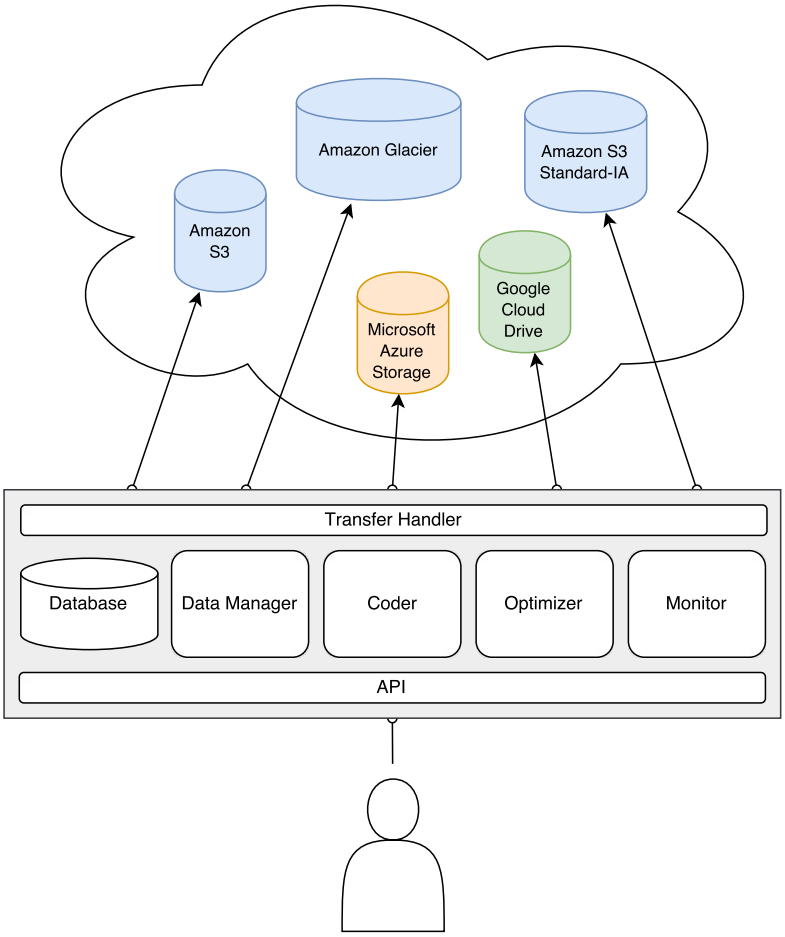


Figure 4.2: Architectural Overview of CORA

storage middleware perfectly. Therefore, we will use CORA as the middleware in our system. We will use it as foundation to integrate the designed optimization algorithms.

4.1.2 CORA

Figure 4.2 shows the system architecture of the middleware CORA. CORA consists of the following parts:

API CORA provides an API that can be used by the user to interact with the middleware. The user can read, write or upload data objects by calling the corresponding endpoints of the API. After receiving commands, the API forwards these commands

to the Data Manager which is responsible for further actions that include processing the requests from the user.

Data Manager The Data Manager is the central component of CORA. It coordinates the communication between the different components and organizes the processing of data objects from and to the various cloud storages.

Coder CORA uses erasure coding for the redundant storage of data objects among multiple cloud storages. The Coder transforms the data by applying erasure coding with the defined encoding parameters m and n. For writing data objects to the cloud storages, the Coder splits data objects into multiple data object chunks of equal size. For the read process from the cloud storages, the Coder combines the data object chunks together into complete data objects. During this process, CORA does not read all data object chunks, but only reads the amount of chunks that is needed to successfully restore a data object, which leads to decreased read cost. Moreover, the Coder recreates data objects chunks if it detects faulty chunks or unavailable cloud storages by using the remaining chunks.

Optimizer The goal of using CORA is to optimize the data object placement in a cost-efficient way while considering user-defined SLOs for the data objects. The component that is responsible for that is the Optimizer. The Optimizer contains the optimization algorithm that is discussed later on in this section. Each time an interaction with a data object (i.e., read, write or update) takes place, the Optimizer is executed. Additionally, it can also be executed in predefined optimization intervals. The execution of the Optimizer processes the integrated optimization algorithm that aims at improving the placement of the data objects. Furthermore, the optimization algorithm ensures that the user-defined SLOs are not violated.

Monitor This component monitors every read, write and update action that is performed by the user on an arbitrary data object. By monitoring all system actions, CORA is able to optimize the placement of data objects based on the history of past data object accesses.

Database For monitoring the usage pattern of data objects, CORA stores historical access and storage information of each data object in a local Database. The stored information is used by the Optimizer to predict future usage of data object chunks. The prediction is based on the assumption that the usage pattern of a data object chunk stays the same in the nearer future [31, 29].

Transfer Handler Finally, the Transfer Handler is the component that acts as connector between CORA and the cloud storage providers. That is, the Transfer Handler sends data object chunks to the different cloud storages and receives data object chunks upon read requests from the storages.

4.2 System Model

In this section we introduce the system model. We will define the relevant terms and variables and discuss the optimization model in the following. A detailed overview of all variables including descriptions can be found in Appendix A.

4.2.1 Variables

We define the set of available cloud storages as S, where $s \in S = \{s_1, s_2, \ldots\}$ is a specific cloud storage. The set of all data objects is labeled with F, where $f \in F = \{f_1, f_2, \ldots\}$ indicates a specific data object. Furthermore, each data object f is split into a set of data object chunks K_f, where $k \in K_f = \{k_{f1}, k_{f2}, \ldots\}$ indicates a specific data object chunk. Each data object chunk k_{fi} belongs to exactly one data object f. Moreover, $|S| \geq |K_f|$ holds for every $f \in F$.

The amount of data object chunks n, i.e., the size of K_f, depends on the applied erasure coding configuration. For instance, with a $(2, 3)$ erasure code, each data object is split into $n = 3$ data object chunks and is distributed among three storages. For the reconstruction of the data object, only $m = 2$ data object chunks are needed (i.e., only two cloud storages have to be available at the same time).

As already mentioned in Section 2.2.3, many cloud storage providers offer a Block Rate Pricing model. We incorporate this model into our optimization by taking into account multiple price steps for the traffic cost as well as for the storage cost. In our optimization model, the set of all price steps of the storage cost for storage s is labeled with B_s^{st}, where $b_s = (b_s^L, b_s^U, p_s) \in B_s^{st} = \{b_{s1}, b_{s2}, \ldots\}$ defines one price step. The variable b_s^L defines the lower bound of the price step, the variable b_s^U defines the upper bound of the price step and p_s defines the actual price for this step. Analogously to the storage cost, $B_s^{T_{out}}$ is the set of all price steps for the outgoing data transfer cost. Since the majority of cloud storage providers do not charge for incoming data transfer, we can neglect the formulation of incoming data transfer prices at this point.

4.2.2 Cost Model

The total cost that are billed to store a data object chunk k on a cloud storage s is composed of three different parts; storage cost (i.e., cost for the amount of needed storage space), request cost (i.e., cost per data access operation) and data transfer cost (i.e., cost per amount of transferred data). Additionally, there may also incur cost for the migration between cloud storages. If a data object chunk is stored on a long-term storage, we also have to consider the BTU and BSU.

Total Cost

$$c_{(s,k,\tau)} = c^S_{(s,k,\tau)} + c^R_{(s,k,\tau)} + c^W_{(s,k,\tau)} + c^{T_{in}}_{(s,k,\tau)} + c^{T_{out}}_{(s,k,\tau)} \tag{4.1}$$

The calculation of the total cost is shown in (4.1). This equation calculates the total cost c for storing a data object chunk k on cloud storage s by taking the usage history of the last τ minutes of k into account.

Storage Cost

$$c^S_{(s,k,\tau)} = p^S_{(s,\gamma_{(s,k)})} \cdot (\sigma_{(k,\tau)} + \hat{\sigma}_{(k,BTU)} \cdot h_k) \tag{4.2}$$

The equation to calculate the storage cost is shown in (4.2). The term $p^S_{(s,\gamma_{(s,k)})}$ calculates the storage price for storing data object chunk k on cloud storage s. Since most cloud storage providers offer a price reduction based on the amount of used storage space, we have to consider this in the price calculation. To incorporate this price reduction, we use $\gamma_{(s,k)}$ to calculate the used storage space in the current billing period (e.g., the current month). If a data object chunk k is currently not stored on cloud storage s, k is added to the currently used storage space $\gamma_{(s,k)}$. The calculated storage price is multiplied with the size of data object chunk k, labeled as $\sigma_{(k,\tau)}$, by considering the last τ minutes. If cloud storage s has defined a BSU and the size of k is smaller than the BSU, $\sigma_{(k,\tau)}$ returns the value of the BSU.

If data object chunk k is currently stored on a long-term storage, we further have to add the storage cost for the remaining time until the BTU is reached. We integrate this with the term $\hat{\sigma}_{(k,BTU)} \cdot h_k$ into the price calculation. $\hat{\sigma}_{(k,BTU)}$ returns the size of data object chunk k, which is charged for the remaining time from now until the end of the BTU. The variable $h_k \in \{0, 1\}$ defines if data object chunk k is currently stored on a long-term storage ($h_k = 1$) or on a standard storage ($h_k = 0$).

Request Cost

$$c^R_{(s,f,\tau)} = r^R_{(k,\tau)} \cdot p^R_s \tag{4.3}$$

$$c^W_{(s,f,\tau)} = r^W_{(k,\tau)} \cdot p^W_s \tag{4.4}$$

The read cost calculation (i.e., cost incurred by read operations) is shown in (4.3). The value $r^R_{(k,\tau)}$ expresses the number of read requests that were performed on data object chunk k in the last τ minutes. p^R_s defines the price per read operation.

The write cost calculation, shown in (4.4), is determined analogously to (4.3).

Since most of the cloud storage providers do not charge for deleting data objects, we ignore the delete cost in our system model. However, the delete cost would also be calculated analogously to (4.3) and (4.4).

Data Transfer Cost

$$c^{T_{out}}_{(s,k,\tau)} = t^{out}_{(k,\tau)} \cdot (p^{T_{out}}_{(s,\beta_{(s,k)})} + p^{ret}_{s} \cdot h_k) \tag{4.5}$$

$$c^{T_{in}}_{(s,k,\tau)} = t^{in}_{(k,\tau)} \cdot p^{T_{in}}_{(s,\beta_{(s,k)})} \tag{4.6}$$

The data transfer cost for outgoing data is shown in (4.5). $t^{out}_{(k,\tau)}$ defines the amount of bytes that were read from the cloud storage provider during time period τ. For instance, if a data object chunk k of size 1MB is read five times during τ, then $t^{out}_{(k,\tau)} = 5 \cdot 1MB = 5MB$. $p^{T_{out}}_{(s,\beta_{(s,k)})}$ is the outgoing data transfer price for the cloud storage s. Analogous to the storage cost, most cloud storage providers offer a price reduction for greater amount of transferred data. Similar to $\gamma_{(s,k)}$ in (4.2), $\beta_{(s,k)}$ calculates the amount of transferred data of cloud storage s in the current billing period. This calculation includes the amount of transferred data of data object chunk k. If cloud storage s is a long-term storage, there may be additional retrieval cost that are charged by the provider. This retrieval cost is determined by $p^{ret}_{s} \cdot h_k$, where p^{ret}_{s} is the data retrieval price. h_k is the same as in (4.2).

The cost for incoming data, shown in (4.6), is determined analogously to (4.5), with the exception that we do not have to consider data retrieval cost for incoming data.

Migration Cost

Up to this point, all discussed cost calculations, namely Equations (4.2) to (4.6), are included in the calculation of the total cost (4.1). The migration cost, shown in (4.7) and (4.8), will be taken into account in Section 4.3, where we will formulate the objective function of the optimization problem.

$$c^{M}_{(s_1,s_2,k)} = (p^{T_{out}}_{(s_1,\beta_{(s_1,k)})} + p^{T_{in}}_{(s_2,\beta_{(s_2,k)})} + p^{ret}_{s_1} \cdot h_k) \cdot \hat{\sigma}_k + r^{R}_{s_1} \cdot p^{R}_{s_1} + r^{W}_{s_2} \cdot p^{W}_{s_2} \tag{4.7}$$

$$c^{M_{red}}_{(s_1,s_2,k)} = (p^{T_{(out,red)}}_{(s_1,\beta_{(s_1,k)})} + p^{T_{(in,red)}}_{(s_2,\beta_{(s_2,k)})} + p^{ret}_{s_1} \cdot h_k) \cdot \hat{\sigma}_k + r^{R}_{s_1} \cdot p^{R}_{s_1} + r^{W}_{s_2} \cdot p^{W}_{s_2} \tag{4.8}$$

The migration cost (i.e., the cost for transferring data from one cloud storage to another) depend on the providers of the source and destination storage. The first case, where the source and destination providers are different, is shown in (4.7). In this case, the migration price is composed of the outgoing price of the source storage s_1, labeled as $p^{T_{out}}_{(s_1,\beta_{(s_1,k)})}$, and the incoming price of the destination storage s_2, labeled as $p^{T_{in}}_{(s_2,\beta_{(s_2,k)})}$. These two terms are the same as in (4.5) and (4.6), respectively. If the source storage s_1 is a long-term storage, we also have to add the retrieval price to the migration price, which is done by the term $p^{ret}_{s_1} \cdot h_k$, where the variables are the same as in (4.5). The migration price is multiplied with the current size of data object chunk k, defined as $\hat{\sigma}_k$. Furthermore, the amount of required read and write operations have to be included in

the calculation of the migration cost. This is done by the terms $r^R_{s_1} \cdot p^R_{s_1}$ and $r^W_{s_2} \cdot p^W_{s_2}$, where the variables are the same as in (4.3) and (4.4).

If the providers of the source and destination storage are the same, the migration can usually be done at a reduced migration price since most cloud storage providers offer a price reduction for migrating data from one region to another, which is shown in (4.8). The first term, depicted as $p^{T_{(out,red)}}_{(s_1,\beta_{(s_1,k)})}$, is the outgoing migration price from one region to another. The second term, depicted as $p^{T_{(in,red)}}_{(s_2,\beta_{(s_2,k)})}$, is the incoming migration price. All other terms are the same as in (4.7).

4.2.3 Decision Variables

To mark if a data object chunk k is stored on a cloud storage s, we use the binary decision variable $x_{(s,k)} \in \{0,1\}$. If k is stored on s, then $x_{(s,k)} = 1$, otherwise $x_{(s,k)} = 0$. As already explained in Section 4.2.2, the decision variable $h_k \in \{0,1\}$ states if data object chunk k is currently stored on a long-term storage. If k is stored on a long-term storage, then $h_k = 1$, otherwise $h_k = 0$. The decision variable $\hat{h_s}$ states if storage s is a long-term storage. If s is a long-term storage, then $\hat{h_s} = 1$, otherwise (i.e., if s is a standard storage) $\hat{h_s} = 0$. The decision variable $z_{(s_1,s_2)} \in \{0,1\}$ states if two storages s_1 and s_2 have different storage providers. If s_1 and s_2 have different providers, then $z_{(s_1,s_2)} = 1$, otherwise (i.e., if they have the same provider) $z_{(s_1,s_2)} = 0$. Similarly, the decision variable $y_{(s_1,s_2)} \in \{0,1\}$ states if two storages s_1 and s_2 have the same storage provider. If s_1 and s_2 have the same provider, then $y_{(s_1,s_2)} = 1$, otherwise (i.e., if they have different providers) $y_{(s_1,s_2)} = 0$. Both decision variables $z_{(s_1,s_2)}$ and $y_{(s_1,s_2)}$ result in 0 if the cloud storages s_1 and s_2 are identical.

Furthermore, the decision variable $g_{(\tilde{S},f)} \in \{0,1\}$ states if every cloud storage $s \in \tilde{S}$ has a data object chunk $k \in K_f$ stored. If every cloud storage $s \in \tilde{S}$ has exactly one data object chunk $k \in K_f$ stored, then $g_{(\tilde{S},f)} = 1$, otherwise (i.e., if at least one of the storages does not have a data object chunk of K_f stored) $g_{(\tilde{S},f)} = 0$.

The decision variables $u^{st}_{(s,b_s)} \in \{0,1\}$, $v^{st}_{(s,b_s)} \in \{0,1\}$ and $o^{st}_{(s,b_s)} \in \{0,1\}$ are used to specify if the total used storage space of storage s is between an upper and a lower bound of the Block Rate Pricing model. If the used storage space of storage s is greater than the lower bound b^L_s, as defined in b_s, then $u^{st}_{(s,b_s)} = 1$, otherwise (i.e., the used storage space of storage s is lower or equal to the lower bound b^L_s) $u^{st}_{(s,b_s)} = 0$. If the used storage space of storage s is lower than the upper bound b^U_s, as defined in b_s, then $v^{st}_{(s,b_s)} = 1$, otherwise (i.e., the used storage space of storage s is greater or equal to the upper bound b^U_s) $v^{st}_{(s,b_s)} = 0$. Furthermore, $o^{st}_{(s,b_s)} = 1$ indicates that the used storage space is between the lower and the upper bound of b_s, which means that both $u^{st}_{(s,b_s)} = 1$ and $v^{st}_{(s,b_s)} = 1$ hold. In the same manner, the decision variables $u^{T_{out}}_{(s,b_s)} \in \{0,1\}$, $v^{T_{out}}_{(s,b_s)} \in \{0,1\}$ and $o^{T_{out}}_{(s,b_s)} \in \{0,1\}$ are defined for the outgoing data transfer price.

4.3 Global Optimization

After defining the system model, including all relevant terms and variables, we are now able to formulate the optimization problem. As already explained previously, the optimization problem that we are going to solve in this thesis is the problem of data object placement on cloud storages with the aim of minimizing total storage cost, while taking into account user-defined QoS constraints.

4.3.1 Objective Function

The objective function of the optimization problem, shown in (4.9), is defined as follows:

$$\begin{aligned}\text{minimize} \sum_{s \in S} \Bigg(\sum_{f \in F} \sum_{k \in K_f} \Big(& c^{R}_{(s,k,\tau)} + c^{W}_{(s,k,\tau)} + c^{M}_{(\hat{s}_k,s,k)} \cdot z_{(\hat{s}_k,s)} \\ & + c^{M_{red}}_{(\hat{s}_k,s,k)} \cdot y_{(\hat{s}_k,s)} \Big) \cdot x_{(s,k)} + c^{st}_{(s,\tau)} + c^{T_{out}}_{(s,\tau)} \Bigg)\end{aligned} \tag{4.9}$$

The objective function is a triple sum that iterates over all cloud storages $s \in S$, over all data objects $f \in F$ and over all data object chunks $k \in K_f$. The first two terms, $c^{R}_{(s,k,\tau)}$ and $c^{W}_{(s,k,\tau)}$, calculate the read and write request cost, as discussed in Section 4.2.2. The terms $c^{M}_{(\hat{s}_k,s,k)} \cdot z_{(\hat{s}_k,s)}$ and $c^{M_{red}}_{(\hat{s}_k,s,k)} \cdot y_{(\hat{s}_k,s)}$ calculate the migration cost from one cloud storage to another. If the source and destination providers are different, the first term, i.e., $c^{M}_{(\hat{s}_k,s,k)} \cdot z_{(\hat{s}_k,s)}$, is applied. Otherwise (i.e., if the source and destination providers are the same), the second term, i.e., $c^{M_{red}}_{(\hat{s}_k,s,k)} \cdot y_{(\hat{s}_k,s)}$, is applied. This whole term is multiplied with the decision variable $x_{(s,k)}$, which states if data object chunk k is stored on cloud storage s or not.

Due to the fact that the global optimization optimizes the placement of all data objects in every step, there exists the possibility that all data objects of a particular storage change during the optimization. Since we consider Block Rate Pricing models in our optimization, this may lead to a change of the current pricing step which could increase or decrease the total cost. Therefore, we split up the calculation of the total cost in two parts. In one part we include all cost calculations without a Block Rate Pricing model. These cost calculations, i.e., read request cost, write request cost and migration cost, were just discussed with the explanation of the objective function. In the other part, we consider the cost calculations that make use of a Block Rate Pricing model. We model these cost calculations as constraints. We will now discuss these cost constraints in the following section.

4.3.2 Cost Constraints

The second part of the total cost calculation of the objective function, i.e., the storage cost $c^{st}_{(s,\tau)}$ and the outgoing data transfer cost $c^{T_{out}}_{(s,\tau)}$, are calculated by Equation (4.15)

and Equation (4.21), respectively.

Storage Cost

Equations (4.10) to (4.14) specify if the totally used storage space of storage s is in the range of a price step of the pricing model of this storage.

$$\forall s \in S,\ \forall b_s \in B_s^{st},\ \exists b_s^L \in b_s: \quad b_s^L \leq \sum_{f \in F} \sum_{k \in K_f} \sigma_{(k,\tau)} \cdot x_{(s,k)} + M \cdot (1 - u_{(s,b_s)}^{st}) \tag{4.10}$$

$$\forall s \in S,\ \forall b_s \in B_s^{st},\ \exists b_s^L \in b_s: \quad b_s^L > \sum_{f \in F} \sum_{k \in K_f} \sigma_{(k,\tau)} \cdot x_{(s,k)} - M \cdot u_{(s,b_s)}^{st} \tag{4.11}$$

Equations (4.10) and (4.11) define if the total used storage space of storage s is greater than the lower bound b_s^L of a particular price step $b_s \in B_s^{st}$. If this holds true, then $u_{(s,b_s)}^{st} = 1$, otherwise $u_{(s,b_s)}^{st} = 0$. M is a sufficiently large constant that is at least the largest possible value of $\sigma_{(k,\tau)}$, at least b_s^L and at least b_s^U.

$$\forall s \in S,\ \forall b_s \in B_s^{st},\ \exists b_s^U \in b_s: \quad \sum_{f \in F} \sum_{k \in K_f} \sigma_{(k,\tau)} \cdot x_{(s,k)} \leq b_s^U + M \cdot (1 - v_{(s,b_s)}^{st}) \tag{4.12}$$

$$\forall s \in S,\ \forall b_s \in B_s^{st},\ \exists b_s^U \in b_s: \quad \sum_{f \in F} \sum_{k \in K_f} \sigma_{(k,\tau)} \cdot x_{(s,k)} > b_s^U - M \cdot v_{(s,b_s)}^{st} \tag{4.13}$$

Equations (4.12) and (4.13) indicate if the total used storage space of storage s is lower than the upper bound b_s^U of a particular price step $b_s \in B_s^{st}$. If this holds true, then $v_{(s,b_s)}^{st} = 1$, otherwise $v_{(s,b_s)}^{st} = 0$.

$$\forall s \in S,\ \forall b_s \in B_s^{st}: \quad 0 \leq u_{(s,b_s)}^{st} + v_{(s,b_s)}^{st} - 2 \cdot o_{(s,b_s)}^{st} \leq 1 \tag{4.14}$$

Equation (4.14) states if the total used storage space of storage s is between the lower and the upper bound of a price step b_s. This is the case, if $u_{(s,b_s)}^{st} = 1$ and $v_{(s,b_s)}^{st} = 1$. If this holds true, then also $o_{(s,b_s)}^{st} = 1$, otherwise (i.e., if $u_{(s,b_s)}^{st} = 0$ or $v_{(s,b_s)}^{st} = 0$), also $o_{(s,b_s)}^{st} = 0$.

$$\begin{aligned}
&\forall s \in S,\ \forall b_s \in B_s^{st},\ \exists p_s \in b_s: \\
&\sum_{f \in F} \sum_{k \in K_f} p_s \cdot \left(\sigma_{(k,\tau)} + \hat{\sigma}_{(k,\tau)} \cdot (h_k + \hat{h}_s)\right) \cdot x_{(s,k)} - M \cdot (1 - o_{(s,b_s)}^{st}) \leq c_s^{st} \\
&\leq \sum_{f \in F} \sum_{k \in K_f} p_s \cdot \left(\sigma_{(k,\tau)} + \hat{\sigma}_{(k,\tau)} \cdot (h_k + \hat{h}_s)\right) \cdot x_{(s,k)} + M \cdot (1 - o_{(s,b_s)}^{st})
\end{aligned} \tag{4.15}$$

Finally, Equation (4.15) calculates the cost that are charged due to the total used storage space of storage s, labeled as c_s^{st}. The cost is calculated by taking into account the appropriate price p_s of price step $b_s \in B_s^{st}$. In this equation, M is a sufficiently large constant that has to be at least $\sigma_{(k,\tau)} + \hat{\sigma}_{(k,\tau)} \cdot (h_k + \hat{h}_s)$ times the largest possible value of p_s, and at least the largest possible value of c_s^{st}.

Data Transfer Cost

Analogously to the storage cost, Equations (4.16) to (4.20) define if the outgoing data transfer is within the boundaries of a specific price step $b_s \in B_s^{T_{out}}$.

$$\forall s \in S,\ \forall b_s \in B_s^{T_{out}},\ \exists b_s^L \in b_s: \quad b_s^L \leq \sum_{f \in F} \sum_{k \in K_f} t_{(k,\tau)}^{out} \cdot x_{(s,k)} + M \cdot (1 - u_{(s,b_s)}^{T_{out}}) \tag{4.16}$$

$$\forall s \in S,\ \forall b_s \in B_s^{T_{out}},\ \exists b_s^L \in b_s: \quad b_s^L > \sum_{f \in F} \sum_{k \in K_f} t_{(k,\tau)}^{out} \cdot x_{(s,k)} - M \cdot u_{(s,b_s)}^{T_{out}} \tag{4.17}$$

Equations (4.16) and (4.17) define if the total outgoing data transfer of storage s is greater than the lower bound b_s^L of a particular price step $b_s \in B_s^{T_{out}}$.

$$\forall s \in S,\ \forall b_s \in B_s^{T_{out}},\ \exists b_s^U \in b_s: \quad \sum_{f \in F} \sum_{k \in K_f} t_{(k,\tau)}^{out} \cdot x_{(s,k)} \leq b_s^U + M \cdot (1 - v_{(s,b_s)}^{T_{out}}) \tag{4.18}$$

$$\forall s \in S,\ \forall b_s \in B_s^{T_{out}},\ \exists b_s^U \in b_s: \quad \sum_{f \in F} \sum_{k \in K_f} t_{(k,\tau)}^{out} \cdot x_{(s,k)} > b_s^U - M \cdot v_{(s,b_s)}^{T_{out}} \tag{4.19}$$

Equations (4.18) and (4.19) indicate if the total outgoing data transfer of storage s is lower than the upper bound b_s^U of a particular price step $b_s \in B_s^{T_{out}}$.

$$\forall s \in S,\ \forall b_s \in B_s^{T_{out}}: \quad 0 \leq u_{(s,b_s)}^{T_{out}} + v_{(s,b_s)}^{T_{out}} - 2 \cdot o_{(s,b_s)}^{T_{out}} \leq 1 \tag{4.20}$$

Equation (4.20) states if the total outgoing data transfer of storage s is between the lower and the upper bound of a price step b_s.

$$\begin{aligned} &\forall s \in S,\ \forall b_s \in B_s^{T_{out}},\ \exists p_s \in b_s: \\ &\sum_{f \in F} \sum_{k \in K_f} t_{(k,\tau)}^{out} \cdot (p_s + p_s^{ret} \cdot h_k) \cdot x_{(s,k)} - M \cdot (1 - o_{(s,b_s)}^{T_{out}}) \leq c_s^{T_{out}} \\ &\leq \sum_{f \in F} \sum_{k \in K_f} t_{(k,\tau)}^{out} \cdot (p_s + p_s^{ret} \cdot h_k) \cdot x_{(s,k)} + M \cdot (1 - o_{(s,b_s)}^{T_{out}}) \end{aligned} \tag{4.21}$$

Moreover, similar to Equation (4.15), Equation (4.21) calculates the cost that are charged due to the total used outgoing data transfer of storage s, labeled as $c_s^{T_{out}}$. The cost are calculated by taking into consideration the appropriate price p_s of price step $b_s \in B_s^{T_{out}}$.

Boundaries

$$\begin{array}{lll} c_s^{st} \geq 0 & c_s^{T_{out}} \geq 0 & \\ u_{(s,b_s)}^{T_{out}} \in \{0,1\} & v_{(s,b_s)}^{T_{out}} \in \{0,1\} & o_{(s,b_s)}^{T_{out}} \in \{0,1\} \\ u_{(s,b_s)}^{st} \in \{0,1\} & v_{(s,b_s)}^{st} \in \{0,1\} & o_{(s,b_s)}^{st} \in \{0,1\} \end{array} \tag{4.22}$$

As the final part of the cost constraints, we add the boundaries for each variable, as shown in (4.22), in order to ensure that the optimization yields correct results.

4.3.3 QoS Constraints

As we have previously defined in Section 2.5, we consider multiple QoS constraints in our optimization. These constraints are defined by the users of the system as SLOs.

Vendor Lock-in Factor

$$\forall f \in F: \qquad \frac{1}{\sum_{k \in K_f} \sum_{s \in S} x_{(s,k)}} \leq l_f \tag{4.23}$$

The vendor lock-in factor is shown in (4.23). For each data object $f \in F$, the user defines the minimum vendor lock-in factor that has to be fulfilled.

Data Availability

$$\forall f \in F: \qquad \sum_{i=m}^{n} \sum_{\tilde{S}' \in [\tilde{S}]^i} \left(\prod_{s \in \tilde{S}'} a_s \cdot \prod_{s \in \tilde{S} \setminus \tilde{S}'} (1 - a_s) \right) \geq a_f \cdot g_{(\tilde{S},f)} \tag{4.24}$$

The constraint for data availability is shown in (4.24).

$\tilde{S} \subseteq S$ is the subset of S that contains the selected cloud storage providers, hence $\tilde{S} = \{s_1, s_2, \ldots, s_n\}$ with $|\tilde{S}| = |K_f| = n$. That is, every data object $k \in K_f$ is split into n data object chunks and every chunk is stored on one particular cloud storage $s \in \tilde{S}$.

$[\tilde{S}]^i = \{R \subseteq \tilde{S}, |R| = i\}$ is the set of i-subsets of $\tilde{S}$ (i.e., the set of subsets of $\tilde{S}$ with size i).

The term $\sum_{\tilde{S}' \in [\tilde{S}]^i} \left(\prod_{s \in \tilde{S}'} a_s \cdot \prod_{s \in \tilde{S} \setminus \tilde{S}'} (1 - a_s) \right)$ calculates the availability of storage set $\tilde{S}'$, which holds all possible combinations of size i of storage set $\tilde{S}$.

The variable a_s defines the availability of cloud storage s. Hence, this part of the equation determines the probability that there are i simultaneously available cloud storages. In addition to this part, we also have to consider the functionality that our system is able to tolerate up to $n-m$ simultaneous storage failures, assuming that the system uses a (m, n) erasure coding configuration. This functionality is incorporated into the calculation of the system availability by the sum $\sum_{i=m}^{n}$.

The calculated system availability is then compared to the term $a_f \cdot g_{(\tilde{S},f)}$, where a_f denotes the minimum availability of data object f as defined by the user and $g_{(\tilde{S},f)}$ states if each cloud storage $s \in \tilde{S}$ has stored one data object chunk $k \in K_f$.

Data Durability

$$\forall f \in F: \qquad \sum_{i=m}^{n} \sum_{\tilde{S}' \in [\tilde{S}]^i} \left(\prod_{s \in \tilde{S}'} d_s \cdot \prod_{s \in \tilde{S} \setminus \tilde{S}'} (1 - d_s) \right) \geq d_f \cdot g_{(\tilde{S},f)} \tag{4.25}$$

The constraint for data availability is shown in (4.25). Since (4.24) and (4.25) are structured similarly, the description of (4.24) can be applied analogously to (4.25).

4.3.4 Remaining Constraints

After defining the objective function as well as the cost and QoS constraints, we continue by defining the remaining constraints.

$$\forall f \in F: \qquad \sum_{k \in K_f} \sum_{s \in S} x_{(s,k)} \cdot \hat{h}_s \geq n - m \tag{4.26}$$

Constraint (4.26) ensures that at least $n - m$ data object chunks of each data object are stored on long-term storages. This is done since only m data object chunks are accessed when downloading a data object. For uploading data objects, providers usually do not charge any cost. Since the storage cost of long-term storages are significantly lower than the storage cost of standard storages, the total cost can be reduced by storing $n - m$ data object chunks on long-term storages.

$$\forall f \in F: \qquad g_{(\tilde{S},f)} \geq \sum_{s \in \tilde{S}} \sum_{k \in K_f} x_{(s,k)} - (n - 1) \tag{4.27}$$

$$\forall f \in F,\ \forall s \in \tilde{S}: \qquad g_{(\tilde{S},f)} \leq \sum_{k \in K_f} x_{(s,k)} \tag{4.28}$$

Constraints (4.27) and (4.28) define the functionality of the decision variable $g_{(\tilde{S},f)}$. The two constraints together result in the functionality of a conjunction. If each cloud storage $s \in \tilde{S} = \{s_1, \ldots, s_n\}$ has a data object chunk $k \in K_f$ stored, then $g_{(\tilde{S},f)} = 1$, otherwise (i.e., if at least one of the storages does not have a data object chunk of K_f stored) $g_{(\tilde{S},f)} = 0$. Hence, $g_{(\tilde{S},f)} = 1$ for the set of selected cloud storages that each have exactly one data object chunk of f assigned.

$$\forall f \in F: \qquad \sum_{k \in K_f} \sum_{s \in S} x_{(s,k)} = n \tag{4.29}$$

Constraint (4.29) ensures that, for every data object $f \in F$, there exist exactly n assignments from data object chunks $k \in K_f$ to cloud storages $s \in S$. Still, it could be the case that multiple data object chunks of one particular data object get assigned to the same cloud storage. Moreover, it could be the case that one particular data object chunk gets assigned to multiple cloud storages. To prevent this, we further need the following constraints.

$$\forall f \in F,\ \forall k \in K_f : \quad \sum_{s \in S} x_{(s,k)} = 1 \tag{4.30}$$

$$\forall f \in F,\ \forall s \in S : \quad \sum_{k \in K_f} x_{(s,k)} \leq 1 \tag{4.31}$$

$$\forall f \in F,\ \forall s \in S : \quad \sum_{k \in K_f} x_{(s,k)} \geq 0 \tag{4.32}$$

Constraint (4.30) ensures that each data object chunk is exactly assigned once. Furthermore, constraints (4.31) and (4.32) ensure that, for every data object, each storage has at most one chunk of this data object assigned and that the value of $x_{(s,k)}$ is non-negative.

$$\begin{aligned} g_{(\tilde{S},f)} &\in \{0,1\} \qquad z_{(s_1,s_2)} \in \{0,1\} \\ x_{(s,k)} &\in \{0,1\} \qquad y_{(s_1,s_2)} \in \{0,1\} \end{aligned} \tag{4.33}$$

Finally, the constraints in (4.33) ensure that the decision variables are binary, i.e., that they can only be 0 or 1.

4.3.5 Similarities to Other Problems

The objective function, shown in (4.9), resembles a global optimization problem since it aims at minimizing cost for the storage of all data objects. The according local optimization problem (i.e., minimizing the storage cost for one particular data object) was already solved in [49]. The system model introduced by Waibel et al. served as foundation for the design of the global optimization problem in this thesis. The main difference between these two models is the fact that, on the one hand, the local optimization problem aims at finding the optimal placement solution for one particular data object. On the other hand, the global optimization seeks for the optimal placement solution for all data objects at the same time.

Furthermore, our introduced global optimization problem is very similar to the generalization of the already introduced MKP, which we described in Section 2.3.2. The generalization of the MKP is the so-called Generalized Assignment Problem (GAP). The difference to the MKP is that every item yields a profit based on the knapsack it is packed into, whereas in the MKP every item yields a defined profit regardless of which knapsack

it is packed into. The Minimization Version of the Generalized Assignment Problem (MINGAP) is the problem that comes closest to our global optimization problem. It is obtained from the GAP by defining cost $c_{ij} = -p_{ij}$ instead of profit p_{ij} for every item j and every knapsack i [27, 33].

4.4 Heuristic Approach

After presenting the global optimization in the previous section, we introduce the heuristic approach in the following section.

4.4.1 Requirements

The heuristic approach is designed with emphasis on resource effectiveness, while still providing a feasible solution to the optimization problem. Since the global optimization is executed each time a file request (i.e., GET, PUT or DELETE) takes place, computational complexity of the optimization problem increases exponentially with the size of the problem instance. While this is acceptable for small problem instances (i.e., small number of data objects and storages), complexity and resource consumption get too high with bigger instances. Thus, the global optimization is not usable in real-world scenarios where big amounts of data need to be processed. The heuristic approach aims at finding a data object placement as close as possible to the optimal data object placement which is returned by the global optimization. The heuristic approach shall find such a cost-efficient data object placement in a shorter amount of time compared to the global optimization. This is particularly important for being able to handle big amounts of data, thus making the heuristic approach usable for real-world use.

4.4.2 Placement Algorithm

The basic heuristic placement algorithm is shown in Algorithm 4.1 and works as follows. For each request on a data object (i.e., GET, PUT or DELETE) the according function to determine a new cost-efficient placement solution is called. If the request is a PUT request and a data object is uploaded for the first time, then the function *uploadPlacement* is called. This can be seen in Lines 4 and 5. If the request is a GET request, then the function *downloadPlacement* is called. This is shown in Lines 8 and 9. Else, i.e., for updating and deleting data objects, we do not optimize the placement. The functions *uploadPlacement* and *downloadPlacement* are described in more detail in Algorithms 4.2 and 4.3.

Upload Placement

The function that optimizes the data object placement when a data object is uploaded is shown in Algorithm 4.2. The function *uploadPlacement* is called each time a new data object is uploaded. The data object, that is to be optimized, is passed as input parameter f. Assuming that we use an erasure coding configuration of (m, n), we have

Algorithm 4.1: Heuristic Placement Algorithm

```
1  for each request do
2      placement ← ∅;
3      switch requestType do
4          case UPLOAD do
5              placement ← uploadPlacement(newDataObject);
6              break;
7          end
8          case DOWNLOAD do
9              placement ← downloadPlacement(allDataObjects);
10             break;
11         end
12         otherwise do
13             break;
14         end
15     end
16     return placement;
17 end
```

Algorithm 4.2: Upload Placement Function

Input : The new data object to be uploaded f
Output : A list of assignments from cloud storages to data object chunks

```
1  Function uploadPlacement(f)
2      placement ← ∅;
3      storages ← ∅;
4      standardStorages ← getRankedStandardStorages(f);
5      longTermStorages ← getRankedLongTermStorages(f);
6      for i ← 1 to m do
7          storages.add(standardStorages[i]);
8      end
9      for i ← 1 to n − m do
10         storages.add(longTermStorages[i]);
11     end
12     for i ← 1 to n do
13         storage ← storages[i];
14         chunk ← f.chunks[i];
15         placement.add(storage, chunk);
16     end
17     return placement;
18 end
```

m data chunks and $n - m$ parity chunks. Based on the fact that we only need to access m chunks in order to fully reconstruct a data object, we aim at placing m data object chunks on the storages where overall cost for storing and accessing the data is minimal. The remaining $n - m$ chunks do not need to be accessed when downloading a data object. Therefore, we can minimize cost by placing these $n - m$ chunks directly on long-term storages when a new data object is uploaded.

For correctly storing the m data object chunks on the cheapest m storages, we construct a storage ranking for standard storages. In the same way, we also construct a storage ranking for long-term storages. We use long-term storages to store the remaining $n - m$ data object chunks that do not need to be accessed when downloading a data object. However, when a new data object is uploaded, we need to write all n data object chunks and not only the m chunks that are needed to download a data object. Once the two storage rankings are constructed, shown in Lines 4 and 5, we can use them to select the m best ranked standard storages (i.e., according to the storage ranking), which is shown in Lines 6 and 7. Analogously, we select the $n - m$ best ranked long-term storages according to the storage ranking, which is shown in Lines 9 and 10. After selecting the m best ranked standard storages and $n - m$ best ranked long-term storages, we have selected n storages in total. Each of these n storages is then assigned to exactly one data object chunk of data object f, shown in Line 15.

This storage process is based on the fact that long-term storages are particularly designed for storing rarely accessed data objects. Therefore, total cost of storing a mostly unused data object on a long-term storage are significantly lower compared to storing the same data object on a standard storage.

Download Placement

The function that optimizes the data object placement when a data object is downloaded is shown in Algorithm 4.3. To optimize the placement for getting a cost-efficient solution, we particularly aim at optimizing unused data objects. For this, we hold a list of unused data objects and update this list in predefined intervals. We consider a data object as unused if it has not been downloaded in the last τ minutes, i.e. if the total amount of outgoing data transfer in the last time interval of this data object is zero. Every time a download request takes place, we calculate the time difference between the current time and the previous time when a data object was downloaded. If this time difference is above the predefined threshold of $timeStepInterval$, then we update the collection of unused data objects $allUnusedDataObjects$ with the list of unused data objects $unusedDataObjects$ from the current time period. This process is shown from Line 5 to Line 13.

If we reach a $timePeriodCount \geq timePeriodThreshold$, then we optimize all unused data objects from the last $timePeriodThreshold$ time periods, shown in Line 15. The process for the optimization of all unused data objects is shown from Line 16 to Line 28. We iterate over the set of all data objects that were unused in all of the $timePeriodThreshold$

Algorithm 4.3: Download Placement Function

Input : The list of all data objects F
Output : A list of assignments from cloud storages to unused data object chunks

```
1  Function downloadPlacement(F)
2      placement ← ∅;
3      timeDiff ← currentTime − previousTime;
4      if timeDiff ≥ timeStepInterval then
5          unusedDataObjects ← ∅;
6          foreach data object f ∈ F do
7              if isUnused(f) then
8                  unusedDataObjects.add(f);
9              end
10         end
11         allUnusedDataObjects.put(timePeriodCount, unusedDataObjects);
12         timePeriodCount ← timePeriodCount + 1;
13         previousTime ← currentTime;
14     end
15     if timePeriodCount ≥ timePeriodThreshold then
16         foreach unused data object f in all past time periods do
17             for i ← 1 to n do
18                 chunk ← f.chunks[i];
19                 if currentStorage(chunk) is a long-term storage then
20                     do nothing;
21                 else
22                     storage ← getBestRankedLongTermStorage(chunk);
23                     placement.add(storage, chunk);
24                 end
25             end
26         end
27         timePeriodCount ← 0;
28         allUnusedDataObjects ← ∅;
29     end
30     return placement;
31 end
```

past time periods. For every data object of this set, we iterate over all its data object chunks. If a chunk is already placed on a long-term storage, we do nothing. Else, i.e., if a chunk is currently stored on a standard storage, we migrate this chunk to a long-term storage in order to save cost, since the data object is considered rarely used. For this, we select the best ranked long-term storage for the unused chunk, shown in Line 22. The function *getBestRankedLongTermStorage* returns the best ranked long-term storage for a particular data object chunk. For this, the function excludes those long-term storages that already have a chunk of the same data object stored. Therefore, the function returns the best ranked long-term storage besides the storages that are already in use by other chunks of the same data object.

4.5 Latency Consideration

The third approach that we develop during the work on this thesis is an extension of the heuristic placement approach that was discussed in the previous section.

4.5.1 Requirements

Up to this point, the overall focus of the two optimization approaches was clearly defined. We designed the global optimization and the heuristic approach with the aim of cost minimization, where the global optimization always yields the best fitting data object placement. In contrast, the heuristic approach is designed with a second goal in mind, i.e., the heuristic approach should be capable of handling a much bigger amount of data objects, while still providing an acceptable solution in terms of cost minimization. Now, for the latency consideration approach, we also incorporate latencies into the problem of finding the best fitting storage solution. Therefore, the selection of the best fitting storage set does not solely depend on the cost of the storages but also takes into account the access latency of a storage. This means that we accept data object placements that cause a slightly more expensive solution if the solution offers reduced latency. Latency between two endpoints may depend on various properties. Some of the properties might be geographical distance, bandwidth and quality of the used connection or the amount of system load of either endpoint, among others [8].

4.5.2 Placement Algorithm

Since the latency consideration approach is based on the same logic as the heuristic approach, the placement functions that were defined in Section 4.4.2 are applicable in the same manner for this approach. However, for this approach we do not only focus on cost efficiency when selecting the data object placement. Instead, we incorporate measured latencies of the cloud storages into the storage ranking. This ranking is also taken into consideration when a data object is downloaded and when data object chunks are migrated to long-term storages in order to keep latencies as low as possible.

We will go into more detail about the implementation of this approach in Section 5.4.

CHAPTER 5

Implementation

"Start by doing what's necessary; then do what's possible; and suddenly you are doing the impossible."

Francis of Assisi

In this chapter we will present details about the implementation of the previously designed system, which we described in Chapter 4. First, we will state some general details about the implementation of CORA and its components. Then we will describe the transformation of the global optimization problem into code. After this, we will outline how we implemented the storage ranking, which is the core functionality of the heuristic approach. Since we have already presented the functionality of the heuristic approach in the previous chapter, we will focus on the implementation of the storage ranking in this chapter. And finally, we will present the foundation for the latency consideration approach, which is the latency measurement of different storage regions.

5.1 General Implementation Details

The middleware CORA is developed with Java 8[1]. The foundation for the project is given by Spring Boot[2]. As a build tool, the project uses Apache Maven[3]. While CORA can be used as a real-world cloud-based storage middleware, it can also be used to simulate a file access trace by reading in the trace line by line and executing the file requests in the

[1] https://www.java.com/en/download/faq/java8.xml
[2] https://projects.spring.io/spring-boot/
[3] https://maven.apache.org/

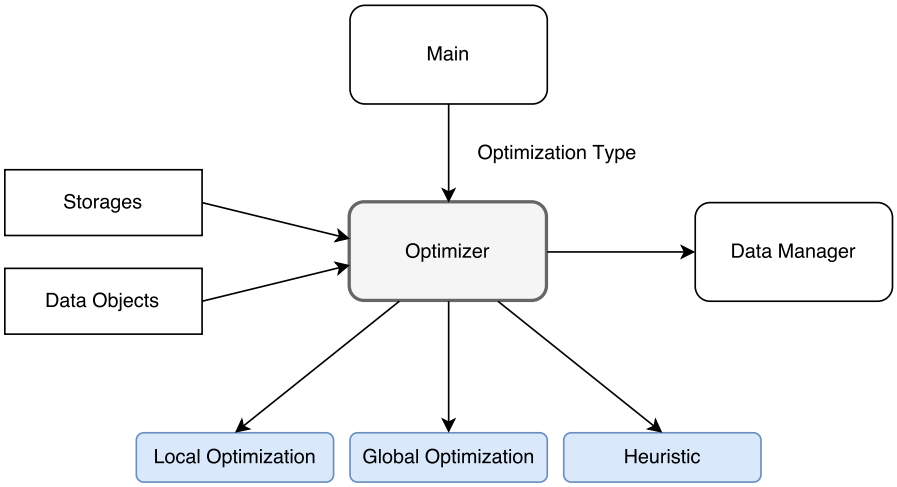


Figure 5.1: Overview of the Placement Selection Functionality

same way as originally stated in the trace. For simulating the exact point in time of every file request, the middleware fakes the current time for each request by setting the system time to the time the request took place originally. For this functionality, CORA uses Joda Time[4]. Another feature of CORA is the ability to split data objects into multiple data object chunks by applying erasure coding. This is realized with the usage of the library Backblaze Reed-Solomon[5], which is an open source implementation of the erasure coding scheme proposed by Reed and Solomon [45].

As has already been described in Section 4.1.2, CORA is designed as a loosely coupled system, which enables the integration of our global optimization model as well as the heuristic approach into the already existing middleware. This enables the possibility to execute different optimization approaches (e.g., local optimization, global optimization, heuristic) by simply changing the optimization class to be used.

An overview of the functionality for the placement selection is given in Figure 5.1. The central component in the selection of the according placement is the *Optimizer*. It takes as input all available cloud storages as well as all stored data objects. Depending on the selected *Optimization Type*, which is set in the *Main* class, the *Optimizer* calls the class with the according optimization logic. The optimization is then executed and a new placement solution is calculated. When the optimization is finished, the optimization result is transferred back to the *Optimizer*. The result of the optimization is passed on as a list of assignments, where each assignment is a mapping from a data object $f \in F$ to a cloud storage $s \in S$. This optimization result is then forwarded by the *Optimizer* to the *Data Manager*, which finally reorganizes the placement of the transferred data objects.

[4] `http://www.joda.org/joda-time/`
[5] `https://www.backblaze.com/open-source-reed-solomon.html`

5.2 Global Optimization

The global optimization integrates the formulation of the according optimization problem, as introduced in Section 4.3, into CORA. For solving the optimization problem we use the commercial software CPLEX[6]. Furthermore, we use the open source library Java ILP[7] to execute CPLEX since it offers an easy-to-use object-oriented Java API.

As code samples of how to transform the formulated optimization problem into Java code in order to execute CPLEX from the code, we give two examples for the implementation of the optimization problem. That is, we detail how we transferred the objective function and one constraint of the global optimization problem into Java code.

Listing 5.1 shows the Java function which constructs the objective function of the global optimization problem, as defined in Section 4.3.1. A `Linear` is a term of an arbitrary amount of sub terms that are all added together. For each sum in the mathematical formulation of the objective function, we have an according `for` loop in the code. In Lines 10-13 we add the read and write cost that arise if we store data object chunk `k` on cloud storage `s`, to `linear`. If we have to migrate the data object chunk to a different storage, then we either add the migration cost for the same provider (see Lines 16-20) or the migration cost for a different provider (see Lines 21-25). In Lines 30-33 we add the data transfer cost for outgoing data as well as the storage cost to `linear`. Finally, in Line 36, we set the minimization of the fully constructed `linear` as the objective of the optimization problem. Note that `linear` in the end contains the sum of the cost of all data objects and all storages in one variable.

In Listing 5.2, the implementation of the vendor lock-in factor, as defined in Section 4.3.3, is shown. The vendor lock-in factor is added to the optimization problem for every data object independently. For each data object `f` in the set of all data objects `F`, we construct a `Linear`, which is the sum over all assignments from cloud storages to data object chunks. Since we cannot formulate terms of linear programming constraints as divisions in CPLEX, we transform the vendor lock-in factor to the form $\sum_{k \in K_f} \sum_{s \in S} x_{(s,k)} \cdot l_f \geq 1$. Therefore, in Line 8, we add the term $l_f \cdot x_{(s,k)}$ to `linear`. After iterating through all data object chunks and all cloud storages, we add `linear` as a constraint to the optimization problem, which is shown in Line 11.

5.3 Heuristic Approach

A key function of the heuristic approach is the ranking of cloud storages in order to determine the cheapest set of cloud storages for every data object. As already mentioned previously, our heuristic approach calculates the best ranked storage set for every data object when it is uploaded in the first place. For generating the storage ranking, we take their pricing models as basis and generate a cost function based on the different cost components.

[6] `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`
[7] `http://javailp.sourceforge.net/`

Listing 5.1: Java Code of the Objective Function

```
1  private void addObjective(Problem problem) {
2      Linear linear = new Linear();
3      for (Storage s : storages) {
4          for (DataObject f : F) {
5              for (DataObjectChunk k : f.getDataObjectChunks()) {
6                  String x_s_k = "x_" + s.getKey() + "_" +
7                          DataObjectChunkUtilities.getDataObjectChunkName(k);
8                  Storage currentStorage =
9                          DataObjectChunkUtilities.getCurrentStorage(k);
10                 double readCost = costCalculation.calcReadCost(s, k);
11                 double writeCost = costCalculation.calcWriteCost(s, k);
12                 linear.add(readCost, x_s_k);
13                 linear.add(writeCost, x_s_k);
14
15                 if (currentStorage != null && currentStorage != s) {
16                     if (currentStorage.getProviderName().equals(
17                             s.getProviderName())) {
18                         linear.add(costCalculation
19                                 .calcMigrationCostSameProvider(
20                                 currentStorage, s, k), x_s_k);
21                     } else {
22                         linear.add(costCalculation
23                                 .calcMigrationCostDifferentProvider(
24                                 currentStorage, s, k), x_s_k);
25                     }
26                 }
27             }
28         }
29
30         String cTout = "cTout_" + s.getKey();
31         linear.add(1, cTout);
32         String cS = "cS_" + s.getKey();
33         linear.add(1, cS);
34     }
35
36     problem.setObjective(linear, OptType.MIN);
37 }
```

Listing 5.2: Java Code of the Vendor Lock-In Factor

```
1  private void addVendorLockInFactor(Problem problem) {
2      for (DataObject f : F) {
3          Linear linear = new Linear();
4          for (DataObjectChunk k : f.getDataObjectChunks().values()) {
5              for (Storage s : storages) {
6                  String x_s_k = "x_" + s.getKey() + "_" +
7                      DataObjectChunkUtilities.getDataObjectChunkName(k);
8                  linear.add(f.getVendorLockInSLA(), x_s_k);
9              }
10         }
11         problem.add(linear, ">=", 1);
12     }
13 }
```

For each cloud storage $s \in S$, we define the cost ranking $r_{(s,k)}$ as follows.

$$r_{(s,k)} = \sigma_k \cdot \frac{p_s^S}{30 \cdot 24} + p_s^W + p_s^R + \sigma_k \cdot (p_s^{T_{out}} + p_s^{ret} \cdot h_k) \tag{5.1}$$

The cost ranking function is shown in (5.1). It is composed of different parts of the pricing model of cloud storage s and the size of data object chunk k, labeled as σ_k. In total, the function value is composed of four terms. The first term, $\sigma_k \cdot \frac{p_s^S}{30 \cdot 24}$, calculates the storage cost for storing data object chunk k on cloud storage s for one hour. We need to divide the storage price p_s^S by $30 \cdot 24$ since p_s^S is defined on a per-month basis. Thus, we get the storage price per hour. The second and the third term calculate the request cost for one write request, labeled as p_s^W, and one read request, labeled as p_s^R, respectively. Finally, the fourth and last term, defined by $\sigma_k \cdot (p_s^{T_{out}} + p_s^{ret} \cdot h_k)$, calculates the data transfer cost for downloading data object chunk k exactly once.

Recall that most of the cloud storages offer a Block Rate Pricing model. In order to facilitate the calculation of the cost function and therefore also the storage ranking, we simplify the calculation by taking as storage price p_s^S the price of the first storage tier (i.e., up to 50TB) for Amazon S3. Since Google only offers one single storage price that is independent of the amount of storage already used, we take this price into account. For the calculation of the outgoing data transfer price we neglect the free tier of Amazon S3 and take the price of the second tier, which ranges from 1GB up to 10TB. For the outgoing data transfer price of Google, we also take the price of the second tier, which ranges from 1TB up to 10TB. Due to the small price difference between the first and the second tier (i.e., 0.12$/GB vs. 0.11$/GB), we can take the price of the second tier without losing much precision in the cost calculation.

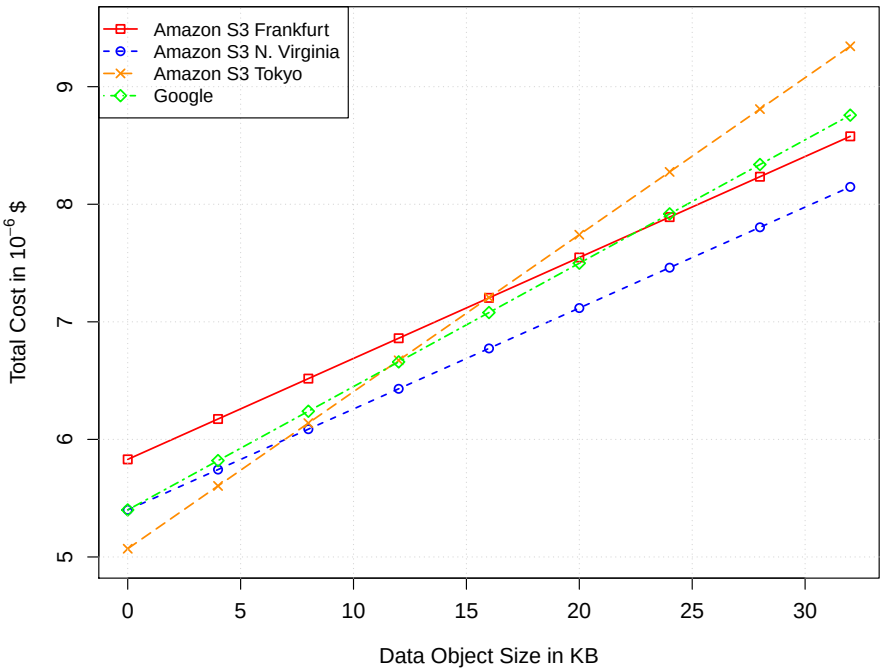


Figure 5.2: Comparison of the Cost of Selected Cloud Storages

Figure 5.2 shows a comparison of the cost of selected cloud storages as a function of the data object size. It can be observed from the graph that Amazon S3 Tokyo is the cheapest storage for small data objects, i.e., for data objects smaller than 7KB. For data objects bigger than 7KB, Amazon S3 Northern Virginia is the cheapest storage according to our defined ranking function. It is also noticeable that Amazon S3 Frankfurt is the most expensive storage among the four shown storages for data object sizes smaller than 16KB. For data object sizes between 16KB and 22KB, it is the third cheapest one and for data object sizes above 22KB, it is the second cheapest storage.

5.4 Latency Consideration

For the optimization approach that considers latency, we assume that the middleware CORA (including our integrated optimization approaches) is deployed in our current location at TU Wien. Based on this location, we measured the latencies of multiple storage regions across the whole globe. For realizing this, we issued read and write requests to selected storages and measured the total execution time of every request. We realized this measurement by uploading files of different sizes (i.e., 1KB, 10KB, 100KB, 1MB, 10MB and 100MB) to selected cloud storages and subsequently downloading these files from the same storages. Since Amazon S3 offers a wide range of storage regions

Table 5.1: Average Upload Times and Standard Deviations in s

File Size	Frankfurt	N. Virginia	N. California	São Paulo	Tokyo
1 KB	0.2 ($\sigma = 0.3$)	0.3 ($\sigma = 0.2$)	0.5 ($\sigma = 0.2$)	0.6 ($\sigma = 0.3$)	0.7 ($\sigma = 0.3$)
10 KB	0.1 ($\sigma = 0.08$)	0.4 ($\sigma = 0.07$)	0.6 ($\sigma = 0.009$)	0.8 ($\sigma = 0.04$)	0.9 ($\sigma = 0.02$)
100 KB	0.2 ($\sigma = 0.04$)	1.2 ($\sigma = 0.3$)	1.9 ($\sigma = 0.3$)	4.4 ($\sigma = 0.9$)	5.5 ($\sigma = 0.2$)
1 MB	0.4 ($\sigma = 0.04$)	2.1 ($\sigma = 0.4$)	3.1 ($\sigma = 0.06$)	8.1 ($\sigma = 1.3$)	8.8 ($\sigma = 1.3$)
10 MB	2.0 ($\sigma = 1.8$)	5.0 ($\sigma = 0.5$)	8.0 ($\sigma = 1.0$)	14.5 ($\sigma = 2.3$)	14.4 ($\sigma = 1.2$)
100 MB	11.1 ($\sigma = 0.7$)	28.4 ($\sigma = 6.2$)	39.5 ($\sigma = 6.5$)	44.4 ($\sigma = 5.8$)	54.5 ($\sigma = 9.5$)

Table 5.2: Average Download Times and Standard Deviations in s

File Size	Frankfurt	N. Virginia	N. California	São Paulo	Tokyo
1 KB	0.06 ($\sigma = 0.04$)	0.1 ($\sigma = 0.02$)	0.2 ($\sigma = 0.02$)	0.3 ($\sigma = 0.03$)	0.3 ($\sigma = 0.02$)
10 KB	0.1 ($\sigma = 0.3$)	0.1 ($\sigma = 0.02$)	0.2 ($\sigma = 0.003$)	0.2 ($\sigma = 0.003$)	0.3 ($\sigma = 0.02$)
100 KB	0.2 ($\sigma = 0.008$)	0.6 ($\sigma = 0.03$)	1.0 ($\sigma = 0.01$)	0.2 ($\sigma = 0.005$)	1.4 ($\sigma = 0.2$)
1 MB	0.2 ($\sigma = 0.1$)	0.6 ($\sigma = 0.03$)	0.9 ($\sigma = 0.01$)	0.2 ($\sigma = 0.003$)	1.3 ($\sigma = 0.04$)
10 MB	0.2 ($\sigma = 0.03$)	0.6 ($\sigma = 0.01$)	0.9 ($\sigma = 0.02$)	0.2 ($\sigma = 0.005$)	1.4 ($\sigma = 0.05$)
100 MB	0.2 ($\sigma = 0.05$)	0.7 ($\sigma = 0.04$)	1.0 ($\sigma = 0.02$)	0.2 ($\sigma = 0.009$)	1.4 ($\sigma = 0.04$)

around the whole globe, we used its Java SDK[8] to measure the latencies (i.e., the total execution time from issuing a read or write request until receiving the response). We repeated the upload and download process of each file for every cloud storage for a total amount of 12 times at different times of the day. Based on these measurements, we calculated the average upload latency (i.e., total execution time for uploading a file) and download latency (i.e., total execution time for downloading a file) for each storage region.

Table 5.1 shows the measurement results for the upload times of different storage regions. The measurement results for the download times of different storage regions are shown in Table 5.2.

As can be observed from Figure 5.3, the upload time of all selected cloud storages is nearly the same for small file sizes of 1KB and 10KB. Up to a file size of 1MB, all upload times are smaller than 10s, while the three storage regions Frankfurt, Northern Virginia and Northern California yield substantially better results than the other two regions, i.e., Tokyo and São Paulo. For big file sizes (i.e., 10MB and 100MB), the results are as expected. That is, the closest region, Frankfurt, is the region with the lowest latency, while the second closest region, Northern Virginia, yields the second lowest latency, etc.

In contrast to the upload time, the download time, shown in Figure 5.4, shows rather balanced results among the selected storage regions. Up to a file size of 10KB, all

[8] `https://aws.amazon.com/sdk-for-java/`

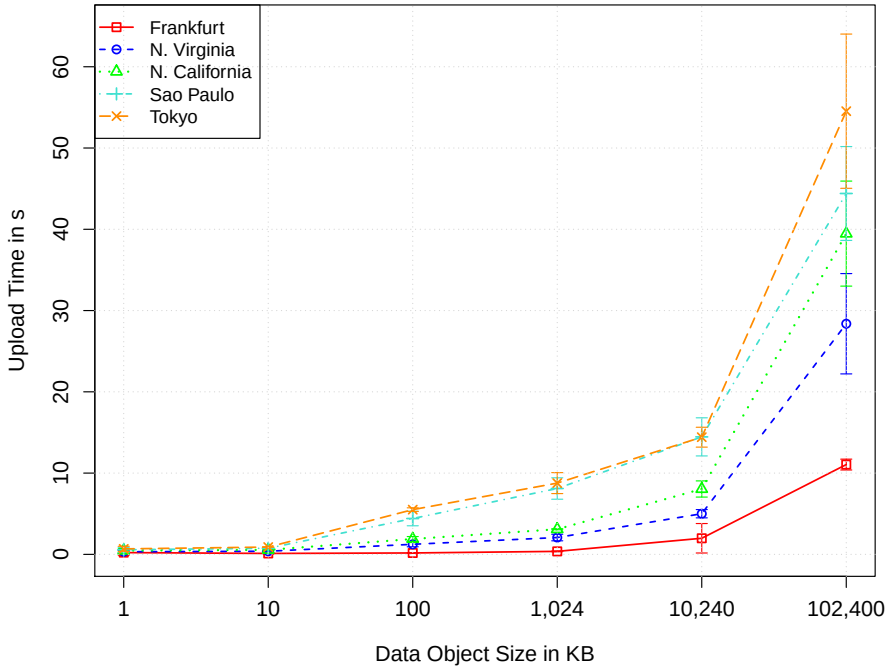


Figure 5.3: Comparison of the Upload Times of Different Storage Regions

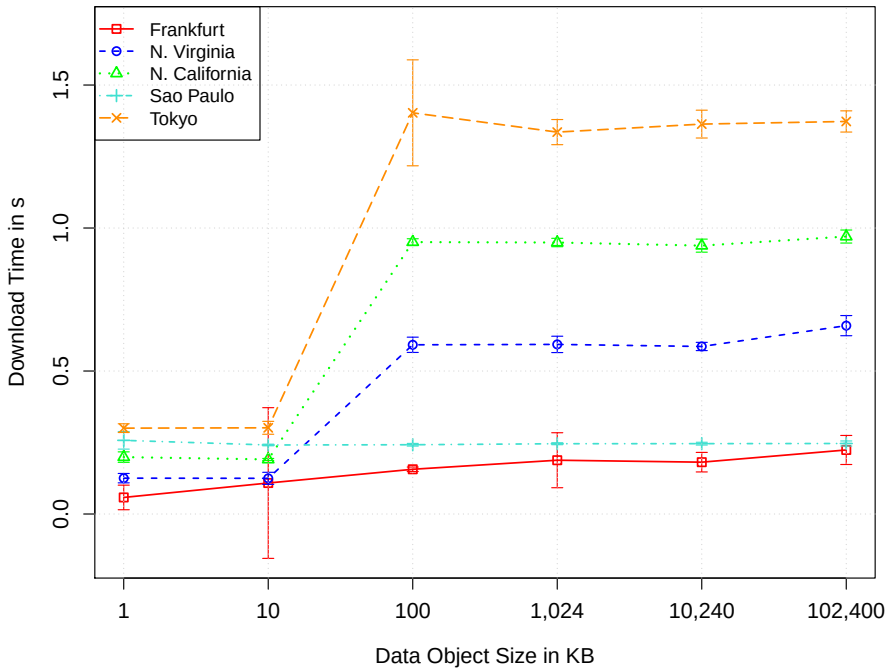


Figure 5.4: Comparison of the Download Times of Different Storage Regions

download times of the selected regions are within close range. For file sizes from 100KB up to 100MB, the results reflect the expected behavior, i.e., Frankfurt is the region with the lowest download latency. However, São Paulo unexpectedly is the region with the second lowest latency, with only small difference to the latency of Frankfurt. Also both the US East (Northern Virginia) and US West (Northern California) regions yield download times of less than 1s. The remaining storage region (Tokyo) shows increased latency with a download time of 1.4s.

In order to incorporate the measured upload and download latencies into the placement selection, we extend the ranking function for the storage ranking from the heuristic approach by adding the access latency into the calculation.

$$r^{L}_{(s,k)} = r_{(s,k)} \cdot (1 + \alpha_{(s,k)}) \tag{5.2}$$

The extended cost ranking with latencies is shown in (5.2). The term $r_{(s,k)}$ is the basic cost ranking from the heuristic approach. We take this cost ranking as basis and multiply the resulting value with $1 + \alpha_{(s,k)}$, where $\alpha_{(s,k)} \in [0, 1]$ is a weighted multiplier that returns low values for storages with low latencies and high values for storages with high latencies. Therefore, the resulting value of the cost ranking will increase slightly for storages with low latencies, while it will increase more significantly for storages with high latencies. If the latency of storage s is 0, the value of the cost ranking stays the same.

The described cost rankings for the latency consideration and for the heuristic approach are used to get the set of best ranked storages. We use these storage rankings to determine the best ranked cloud storages for a particular data object. More precisely, the functions *getRankedStandardStorages* and *getRankedLongTermStorages* from Algorithm 4.2 return all standard and long-term storages, respectively, sorted by the value of the cost ranking in ascending order. This means that the storage with the lowest cost ranking value will be returned at first position, the value with the second lowest cost ranking value at the second position etc. Furthermore, the function *getBestRankedLongTermStorage* from Algorithm 4.3 returns the long-term storage with the lowest cost ranking value which is not already used for another chunk of the same data object.

We further take the latencies of the different storages into consideration when downloading a data object. That is, we do not download the needed data object chunks from the cheapest storages as we do with the heuristic approach. Instead, we download the chunks from the storages with the lowest latencies in order to be able to retrieve the requested data object in the shortest possible amount of time.

Additionally, data objects are only transferred to long-term storages if the latency of the long-term storage is not worse than the latency of the standard storage. This way, we can ensure that the latency does not get worse during the migration of unused data objects.

CHAPTER 6

Evaluation

"In order to succeed, your desire for success should be greater than your fear of failure."

Bill Cosby

After we explained the design and implementation of our optimization approaches in Chapters 4 and 5, we evaluate the implemented optimization approaches in this chapter. First, we will state some prerequisites about the used input data, used cloud storages as well as the defined parameters and how the baseline for the evaluation is constructed. For showing the behavior of the implemented approaches, we will then describe three different evaluation scenarios and explain the functionality of the optimization approaches for each scenario in detail.

6.1 Prerequisites

In this section we state information that is valid for all evaluation scenarios that are going to be described in the following sections. We describe the applied input data as well as the used cloud storages. Moreover, we define the used parameters and explain how we construct the baseline which serves as basis for the evaluation of our results.

6.1.1 Input Data

For the evaluation of the previously discussed and implemented optimization approaches we use the real-world file access trace discussed in [21]. This trace contains one month (i.e., 30 days) of anonymized file access information from cloud storages that were used by more than 1 million users. The trace contains detailed information about every file

Table 6.1: Used Cloud Storages for the Evaluation

Provider	Region	Storage Type
Amazon	N. Virginia (US East)	Standard
	N. Virginia (US East)	IA
	N. California (US West)	Standard
	Frankfurt (EU)	Standard
	Frankfurt (EU)	IA
	Tokyo (Asia Pacific)	Standard
	São Paulo (South America)	Standard
Google	Europe	Regional
Self-Hosted	TU Wien	Standard Long-Term

access that has happened during the analyzed capture duration. The logged information for every access contains the file identifier, access time, amount of transferred data etc.

Since the file access trace contains a huge amount of data object information, we select a smaller set of data objects to use in our evaluation. We select the used data objects equally among all data objects of the trace in order to include both frequently used data objects and seldom used data objects as well as big and small data objects.

Due to the fact that the trace was captured within running operations, it contains access information about data objects that were uploaded prior to the start of the file access trace. Therefore, we upload all data objects for which the file access trace contains a read operation prior to a write operation at the beginning of the evaluation. This way we can ensure that every data object request can be correctly handled by the optimization.

6.1.2 Cloud Storages

Since we evaluate our implemented optimization approaches against a real-world file access trace, we also use real-world cloud storages in our evaluation. We use a total set of 10 cloud storages to evaluate the functionality of the placement optimizations. An overview of all used storages is given in Table 6.1. Among the used cloud storages are different solutions from Amazon S3, Google Cloud Storage and a self-hosted OpenStack Swift[1] storage.

For the various storages by Amazon S3[2] and Google Cloud Storage[3] we use their current pricing models, as of publishing date of this thesis[4]. For the self-hosted OpenStack Swift

[1] `https://wiki.openstack.org/wiki/Swift`
[2] `https://aws.amazon.com/s3/pricing/`
[3] `https://cloud.google.com/storage/pricing`
[4] Publishing date is 21st April, 2017

storage we apply the pricing model of Amazon S3 Frankfurt, i.e., for the self-hosted standard storage we apply the Amazon S3 Standard (Frankfurt) pricing model and for the self-hosted long-term storage we apply the Amazon S3 IA (Frankfurt) pricing model.

6.1.3 Parameters

As the duration of the file access trace is 30 days, we set the BTU to one week (i.e., 7 days) in order to prove the correct functionality of our solution. We do this due to the fact that we can only show the correct behavior of our optimization approach if the BTU is significantly smaller than the total duration of the evaluation. CORA creates a new time step every 12 hours. This setting has already proved to achieve good results in [49]. For every time step, the middleware logs the performed operations that are done in this particular time period. The global optimization takes the access history of the 5 most recent time steps (i.e., 60 hours) into consideration when determining a new cost-efficient placement solution. For the heuristic approach we set the $timePeriodThreshold$ to 10, which means that a data object is considered unused if its outgoing data transfer in the last 10 time periods is zero. We set this variable to 10 since our detailed evaluations have yielded best results for this setup.

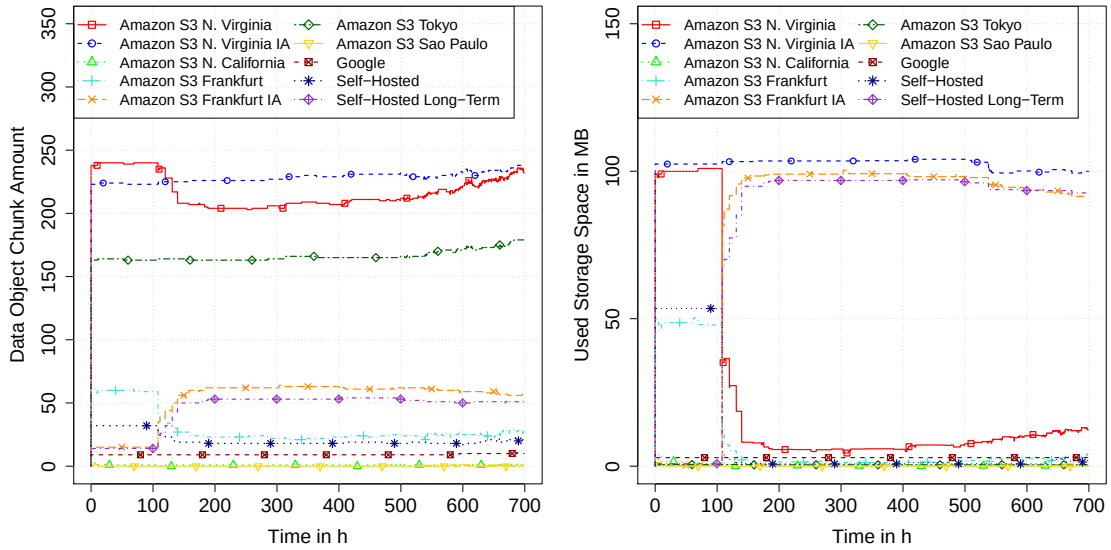
For every data object $f \in F$, we define the SLOs as follows. We set the vendor lock-in factor $l_f = 0.5$, the availability $a_f = 99.99\%$ and the durability $d_f = 99.9999999\%$. Furthermore, we use a $(2, 3)$ erasure coding configuration. This means that every data object is split into 3 data object chunks of equal size and every data object can be reconstructed by downloading 2 of 3 data object chunks. Again, this setting proved to yield good results in [49].

6.1.4 Baseline

For being able to correctly estimate the functionality of our solution, we compare the optimization approaches to a baseline. The baseline shows the cost that would be incurred if all data objects of the file access trace were uploaded to and downloaded from a fixed set of cloud storages. To show the actual advantage of our optimization approaches in terms of cost, we construct the set of cloud storages for the baseline by selecting the three cheapest standard storages. In our case these storages are Amazon S3 Northern Virginia, Amazon S3 Frankfurt and the self-hosted OpenStack Swift storage.

6.2 Small Scale Scenario

The first scenario that we are going to evaluate is a small scale scenario where we compare the global optimization and the heuristic approach to the baseline. We also outline advantages in terms of cost reduction between our developed optimization approaches and the local optimization from [49]. Due to the fact that the complexity of the global optimization problem increases exponentially with the size of the input data, i.e., the amount of data objects and used cloud storages, we use a small selection of 300 data



(a) Data Object Chunk Amount

(b) Used Storage Space

Figure 6.1: Data Object Chunk Distribution of the Global Optimization

objects for this evaluation scenario. Our detailed evaluations have shown that the global optimization is not feasible for greater amounts of data objects. However, we can still show the correct functionality of the global optimization and furthermore evaluate the quality of the heuristic approach in comparison to the local and the global optimization with this small scale scenario.

Evaluation Hypothesis

In the first step of the evaluation all optimization approaches select the cheapest storages to store new data objects. After a while, all optimization approaches are expected to migrate rarely used data object chunks from standard storages to long-term storages. Depending on the optimization interval, the migration starts sooner or later. Since the global optimization optimizes the placement of all data objects every time a data object access takes place, it should transfer rarely used data object chunks to long-term storages every time the optimization discovers a new cost-efficient solution. In contrast, the local optimization and the heuristic approach optimize the placement in predefined intervals. For instance, the heuristic approach transfers unused data object chunks to long-term storages only when they are not used for at least 10 time steps, which in our case means at least 5 days (since CORA creates a new time step every 12 hours). Therefore, the global optimization should yield the best results, i.e., it should be the most economical approach among all developed approaches, since it optimizes the placement of all data objects continuously.

Execution

Figure 6.1 shows the data object chunk distribution of the global optimization. Figure 6.1a shows the amount of data object chunks stored on each storage, while Figure 6.1b shows the used storage space of each storage.

It can be observed from Figure 6.1a that the global optimization uploads two out of three data object chunks of most new data objects to the Amazon S3 Northern Virginia standard storage and the Amazon S3 Northern Virginia IA storage. Most of the third chunks of new data objects are uploaded to the Amazon S3 Tokyo storage. The third chunk of the rest of the data objects is either uploaded to the Amazon S3 Frankfurt standard storage, the self-hosted standard storage, the Amazon S3 Frankfurt IA storage, the self-hosted long-term storage or the Google storage.

Every time a data object access takes place, the optimization recalculates the cheapest data object placement and migrates data object chunks if necessary. The migration of data object chunks from standard storages to long-term storages can be seen in the graph between 100 hours and 200 hours of execution time. At this point some chunks are transferred from the three standard storages Amazon S3 Northern Virginia, Amazon S3 Frankfurt and the self-hosted standard storage to the long-term storages Amazon S3 Frankfurt IA and the self-hosted long-term storage. This can be observed from the graph as the amount of data object chunks on the standard storages decreases, whereas the amount of chunks on the long-term storages increases to the same extent.

If we compare the total amount of data object chunks stored on standard storages with the amount of data object chunks stored on long-term storages, we can see in Figure 6.1a that after the migration there are still more data object chunks stored on standard storages. However, it can be observed from Figure 6.1b that the amount of used storage space is considerably higher on long-term storages. This leads to the conclusion that the global optimization transfers rather bigger data objects to long-term storages and leaves the smaller ones on standard storages. We can explain that by the fact that the bigger data objects account for a bigger portion of the total cost than the smaller data objects do. Furthermore, due to the BSU of long-term storages, which is 128KB for the long-term storages that are used in our evaluation, migrating data object chunks only results in cost savings if the transferred chunks are at least the size of the BSU. Therefore, a lot of cost can be saved by migrating big data objects, i.e., data objects with chunk sizes of at least the BSU.

It can be observed from Figure 6.1b that in the end of the evaluation the Amazon S3 Northern Virginia IA storage, the self-hosted long-term storage and the Amazon S3 Frankfurt IA storage use the greatest amount of storage space. By observing Figure 6.1a we see that the Amazon S3 Northern Virginia IA storage also holds the greatest amount of data object chunks among all storages. Furthermore, the Amazon S3 Northern Virginia standard storage holds the second most data object chunks. However, if we take a look at the used storage space of both storages, we can see that the standard storage does not use a lot of storage space. On the other hand, the long-term storage uses the most

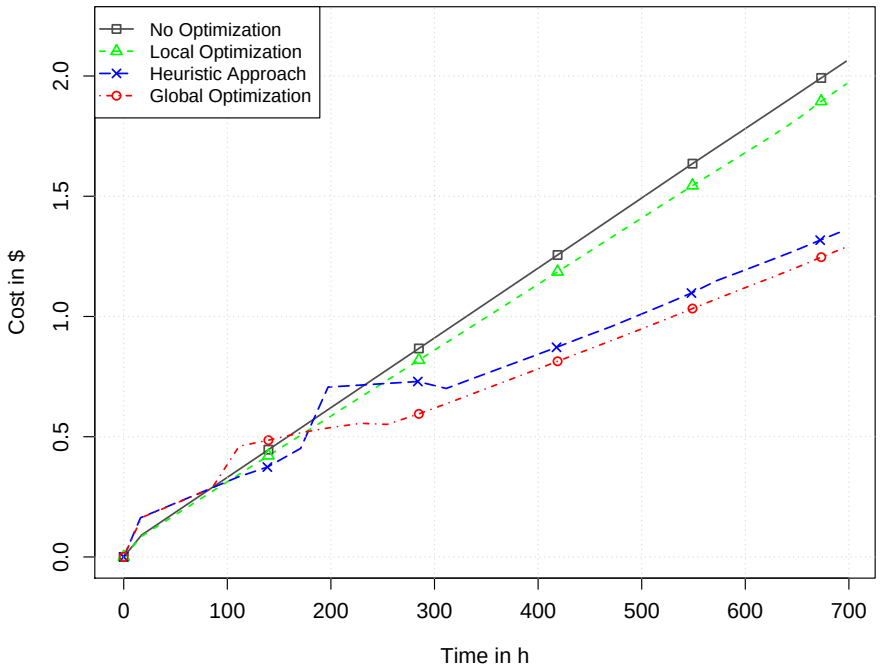


Figure 6.2: Total Cost of the different Optimization Approaches for the Small Scale Evaluation Scenario

storage space among all storages. Thus, we conclude that most of the small data object chunks remain on standard storages, whereas bigger chunks are transferred to long-term storages.

Also noticeable is the fact that the Amazon S3 Tokyo storage holds the third greatest amount of data object chunks. However, the used storage space of this storage is nearly zero, which leads to the conclusion that only very small data object chunks are stored on this storage. This can also be derived from Figure 5.2 which shows that the Amazon S3 Tokyo storage is the cheapest storage for small data objects.

In summary, 42% of all data object chunks are transferred to long-term storages. These 42% use a total amount of storage space of 284MB. The remaining 58% of the chunks are left on standard storages. They only use a total amount of storage space of 22MB. Therefore, the 42% that are migrated to long-term storages incur 93% of the total cost, whereas the remaining 58% that are left on standard storages only incur 7% of the total cost. Since the storage cost of long-term storages are essentially lower than the storage cost of standard storages, the transfer of unused data objects from standard storages to long-term storages results in significant cost savings.

Evaluation Results

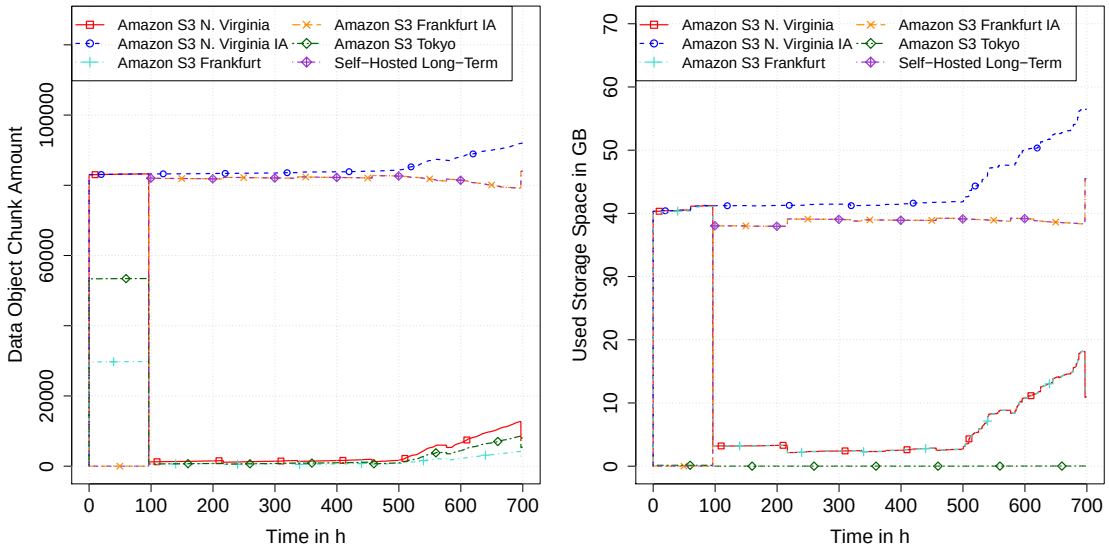
Figure 6.2 shows the cumulative cost of the global optimization in comparison with the local optimization, the heuristic approach and the baseline. It can be seen from the graph that the cost of the global optimization and the heuristic approach increase right from the beginning. This is due to the fact that both approaches use one long-term storage for the storage of new data objects. The BTU of the long-term storages leads to an initial cost increase. However, this cost increase is compensated with a lower gradient of the cost curve. After 90 hours, the global optimization migrates a big amount of data object chunks to long-term storages which also leads to a temporary cost increase due to the BTU cost. After the BTU for the initially uploaded data objects and the migrated data object chunks is over, the cost curve of the global optimization increases with a lower gradient in comparison with the baseline. This can be seen in the graph after 258 hours, i.e., 90 hours (start of migration) + 168 hours (BTU). From this moment on, the benefit of the global optimization compared to the baseline increases more and more. The heuristic approach starts migrating unused data object chunks after 180 hours. This can be derived from the graph since the gradient of the cost curve increases rapidly at 180 hours. After this migration, the cost curve flattens similar to the global optimization. The cost of the heuristic approach also stays below the cost of the baseline until the end of the execution time.

It can be seen from the graph that the local optimization yields small cost savings in comparison with the baseline. In total, the local optimization yields cost savings of 5%. The heuristic approach saves expenses of 34%, and the global optimization saves cost in the amount of 37%.

It can be stated that the cost savings compared to the baseline would even increase more with longer evaluation durations. This is due to the fact that the cost gradient for the approaches that use long-term storages is significantly lower than the gradient of the baseline since the baseline uses a predefined set of standard storages without taking into account long-term storages for rarely used data objects.

6.3 Large Scale Scenario

After evaluating the global optimization and the heuristic approach in the small scale scenario, we now focus on the scalability of the latter. Therefore, we aim at evaluating the heuristic approach in a large scale scenario with 100,000 data objects. As already mentioned previously, the instance size of the global optimization problem increases exponentially with the size of the input data. Moreover, the instance size of the local optimization problem increases linearly with the size of the input data. Due to the big amount of data objects in this scenario, both the local and the global optimization are not feasible anymore. Therefore, we will only compare the heuristic approach to the baseline since we want to show the capability of the heuristic approach to handle big amounts of data objects.



(a) Data Object Chunk Amount

(b) Used Storage Space

Figure 6.3: Data Object Chunk Distribution of the Heuristic Approach

Evaluation Hypothesis

At the start of the evaluation the heuristic approach should upload all data objects to the two best ranked standard storages and the best ranked long-term storage, according to the ranking function defined in Section 5.3. Since the heuristic approach uploads one chunk of every new data object directly to a long-term storage, the cost will rise immediately when the first data objects are uploaded. This is due to the BTU costs that are directly charged for data object chunks stored on long-term storages. After the initial amount of data objects is uploaded, the cost graph will flatten until the heuristic approach migrates unused data object chunks from standard storages to long-term storages. The migration will lead to a temporal cost increase due to the additional BTU costs of the data object chunks that are transferred from standard storages to long-term storages. After the migration of the unused data object chunks is finished, the cost curve will flatten again and rise slower than the baseline. In the end of the evaluation the heuristic approach is expected to incur considerably less cost in comparison to the baseline.

Execution

Figure 6.3 shows the data object chunk distribution of the heuristic approach. Figure 6.3a shows the amount of data object chunks stored on each storage, while Figure 6.3b shows the used storage space of each storage.

We can observe from Figure 6.3a that the heuristic approach distributes newly uploaded data objects among two different sets of cloud storages. That is, depending on the data object size, the heuristic approach selects the two best ranked standard storages as well

as the best ranked long-term storage. This storage ranking is done according to the ranking function defined in Section 5.3. For about two thirds of all new data objects the heuristic approach uses the Amazon S3 Northern Virginia standard storage, the Amazon S3 Northern Virginia IA storage and the Amazon S3 Tokyo storage. For the remaining third of new data objects the heuristic approach uses the Amazon S3 Northern Virginia standard storage, the Amazon S3 Northern Virginia IA storage and the Amazon S3 Frankfurt standard storage.

After 100 hours of execution time the heuristic approach migrates unused data objects from the standard storages Amazon S3 Northern Virginia, Amazon S3 Tokyo and Amazon S3 Frankfurt to the long-term storages Amazon S3 Frankfurt IA and the self-hosted long-term storage. This can be seen in Figure 6.3a as the amount of data object chunks on the three standard storages rapidly decreases while the amount of chunks on the two long-term storages increases. Every 120 hours the heuristic approach starts a new migration of unused data objects. From Figure 6.3b we can observe that some big data object chunks are migrated from the two standard storages Amazon S3 Northern Virginia and Amazon S3 Frankfurt to the two long-term storages Amazon S3 Frankfurt IA and the self-hosted long-term storage after 220 hours.

After 500 hours some big data objects are uploaded. This can be seen from the graphs as the amount of data object chunks on the three standard storages Amazon S3 Northern Virginia, Amazon S3 Frankfurt, Amazon S3 Tokyo and the long-term storage Amazon S3 Northern Virginia IA steadily increases until the end of the execution time. Furthermore, the used storage space of the two former standard storages and the long-term storage also increases steadily. Since the used storage space of the storage Amazon S3 Tokyo does not increase significantly, we can derive that this storage is only used for small data objects. Since the heuristic approach does not transfer any new data objects that are uploaded after 500 hours, we can also state that these data objects are frequently used. Therefore, we will not be able to save a lot of cost compared to the baseline after 500 hours until the end of the execution time.

Evaluation Results

Figure 6.4 shows the cumulative cost of the heuristic approach in comparison with the baseline. We can observe from the graph that the cost of the heuristic approach increases right at the beginning of the evaluation. This is due to the BTU cost that are charged for the storage of data object chunks on the Amazon S3 IA storage. Thus, the heuristic approach is temporarily more expensive than the baseline. After 100 hours, the heuristic approach migrates all unused data object chunks from standard storages to long-term storages. Therefore, the cost curve of the heuristic approach rises again. After the BTU cost are charged, the heuristic approach shows a lower gradient than the baseline. This is due to the fact that the storage cost of long-term storages are significantly lower than the storage cost of standard storages. Since the baseline only uses standard storages, the cost nearly increase linearly from the beginning. In comparison, the heuristic approach

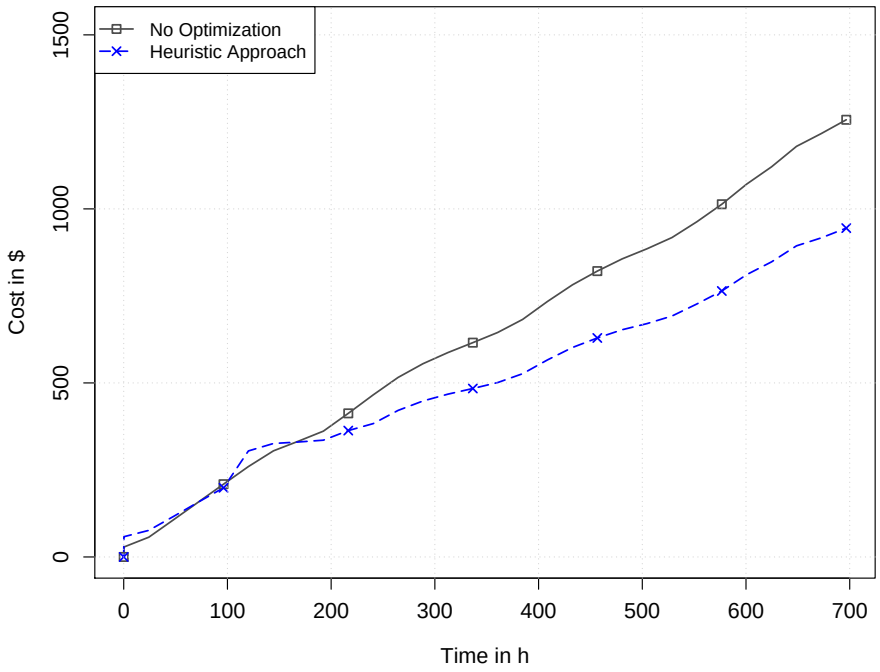


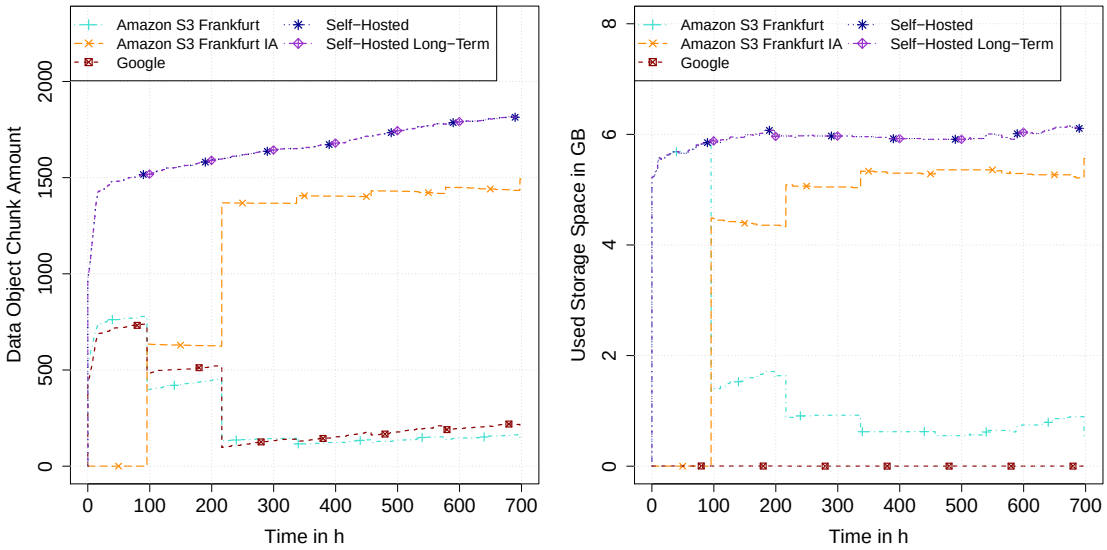
Figure 6.4: Total Cost of the Heuristic Approach for the Large Scale Evaluation Scenario

uses long-term storages for storing unused data object chunks. Therefore, we are able to save an essential amount of cost compared to the baseline.

In summary, we are able to save 25% of total cost by using the heuristic approach instead of the baseline. It has to be added that the cost savings are bigger the more unused big data objects there are. Hence, we could even achieve better results if the amount of big and frequently used data objects was not that big after 550 hours.

6.4 Latency Scenario

In the small and large scale evaluation scenario we evaluated the global optimization and the heuristic approach, respectively, and compared them to the baseline. For this scenario, we show the functionality of the latency consideration approach. We do this by evaluating the latency consideration approach based on two different locations. In the first part of the scenario, we evaluate the implemented approach based on our current location at TU Wien. In the second part of the scenario, we evaluate the latency consideration approach based on the location of Tokyo. By explaining both evaluations in detail, we prove the correct functionality of the implemented approach and we furthermore compare both results against each other. In this scenario we use a total amount of 1,879 data objects from the file access trace. We select both rarely accessed and frequently accessed data objects in order to show the behavior of the latency consideration approach. To



(a) Data Object Chunk Amount

(b) Used Storage Space

Figure 6.5: Data Object Chunk Distribution of the Latency Consideration Approach based on the Location of TU Wien

demonstrate the actual benefit of the latency consideration approach over the heuristic approach, we also compare the latencies of both approaches against each other and show the reduced latencies for these approaches in comparison to the heuristic approach.

6.4.1 TU Wien

For the latency evaluation scenario based on the location of TU Wien, we use the measured latencies as described in Section 5.4. Additionally, we set the latencies of the Google storage to the same latencies as for the Frankfurt region.

Evaluation Hypothesis

In the beginning of the evaluation the optimization should upload all new data objects to the two best ranked standard storages and the best ranked long-term storage, according to the ranking function defined in Section 5.4. Since we evaluate this scenario based on the location of TU Wien, the self-hosted storages are expected to be among the best ranked storages, as they have the lowest latencies among all storages. Also, the Amazon S3 Frankfurt storages and the Google storage should be among the best ranked storages due to their close locations to Vienna.

Execution

Figure 6.5 shows the data object chunk distribution of the latency consideration approach based on the location of TU Wien. Figure 6.5a shows the amount of data object chunks

stored on each storage, while Figure 6.5b shows the used storage space of each storage.

In the beginning of the evaluation we can observe from Figure 6.5a that all new data objects are uploaded to the self-hosted standard storage, the self-hosted long-term storage and either to the Amazon S3 Frankfurt standard storage or the Google storage. As expected, due to their low latencies the self-hosted storages are among the best ranked storages. Depending on the data object size and therefore the resulting storage ranking, chunks of new data objects are either uploaded to the Amazon S3 Frankfurt standard storage or the Google storage. After about 100 hours, unused data object chunks are migrated from both the Amazon S3 Frankfurt standard storage and the Google storage to the Amazon S3 Frankfurt IA storage. After the first migration, a new migration is triggered every 120 hours. After 220 hours, most of the data object chunks from the the Amazon S3 Frankfurt standard storage and the Google storage are migrated to the Amazon S3 Frankfurt IA storage. As most of the data object chunks are migrated to long-term storages, this means that most of these chunks have not been used in the observation period. Note that new data objects are still uploaded to the Amazon S3 Frankfurt standard storage and the Google storage since these two are the best ranked standard storages apart from the self-hosted standard storage which is used anyway due to its low latency.

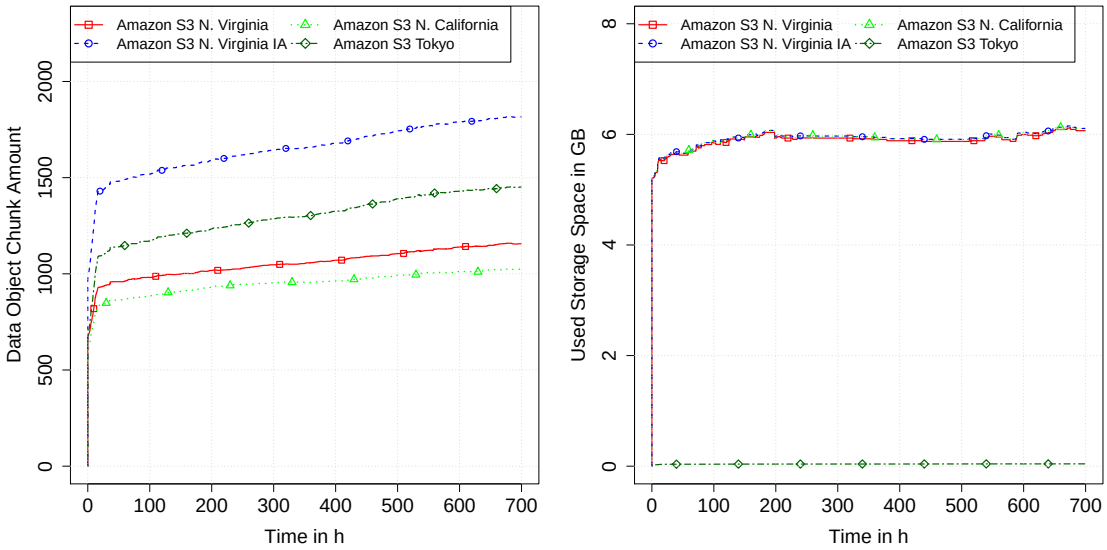
If we consider the used storage space of the different storages in Figure 6.5b, we see that the amount of used storage space for the Google storage is nearly zero. This is due to the fact that the Google storage is cheaper and therefore better ranked than the Amazon S3 Frankfurt standard storage for small data objects. For bigger data objects the Amazon S3 Frankfurt standard storage is better ranked than the Google storage and therefore bigger data objects are uploaded to the Amazon S3 Frankfurt standard storage. Hence, we only see a considerable amount of used storage space for the Amazon S3 Frankfurt standard storage, but not for the Google storage. Besides that, the self-hosted storages use the most amount of storage space among all storages.

Evaluation Results

See Section 6.4.2 for a detailed analysis of the evaluation results for the latency evaluation scenario. There we will compare both parts of the evaluation scenario, i.e., the first part of the scenario based on the location of TU Wien and the second part of the scenario based on the location of Tokyo, against each other.

6.4.2 Tokyo

For the latency evaluation scenario based on the location of Tokyo, we set up an Amazon EC2 instance in the region Asia Pacific (Tokyo) and measured the upload and download times of differently sized data objects between this instance and various cloud storages. We executed this measurement in the same manner as we measured the latencies based on our current location at TU Wien, which is described in Section 5.4. The detailed measurement results can be found in Appendix B. Additionally, we set the latencies of



(a) Data Object Chunk Amount

(b) Used Storage Space

Figure 6.6: Data Object Chunk Distribution of the Latency Consideration Approach based on the Location of Tokyo

the Google storage and the self-hosted storages to the same latencies as for the Frankfurt region.

Evaluation Hypothesis

In the beginning of the evaluation the optimization should upload all new data objects to the two best ranked standard storages and the best ranked long-term storage, according to the ranking function defined in Section 5.4. Since we evaluate this scenario based on the location of Tokyo, the Amazon S3 Tokyo storage is expected to be among the best ranked standard storages. Furthermore, we also expect the Amazon S3 Northern California storage to be among the best ranked standard storages due to its relatively close location to Tokyo, in comparison to the location of the Amazon S3 Northern Virginia storages and the other storages based in Europe. Moreover, we expect the optimization to use the Amazon S3 Northern Virginia IA storage as its best ranked long-term storage, due to the lower latency compared to the location of the Amazon S3 Frankfurt IA storage and the self-hosted long-term storage.

Execution

Figure 6.6 shows the data object chunk distribution of the latency consideration approach based on the location of Tokyo. Figure 6.6a shows the amount of data object chunks stored on each storage, while Figure 6.6b shows the used storage space of each storage.

In the beginning of the evaluation all new data objects are uploaded to the Amazon S3 Northern Virginia IA storage and two standard storages, depending on the data object size. For small data objects, the Amazon S3 Tokyo and Amazon S3 Northern Virginia standard storages are the best ranked standard storages. For medium sized data objects, the optimization selects the Amazon S3 Tokyo and Amazon S3 Northern California standard storages. For big data objects, the Amazon S3 Northern California and Amazon S3 Northern Virginia standard storages are the standard storages with the best ranking. Therefore, the data object chunk distribution shown in Figure 6.6a shows a distribution of data object chunks among the standard storages of Amazon S3 Tokyo, Amazon S3 Northern Virginia and Amazon S3 Northern California. The Amazon S3 Northern Virginia IA storage is the best ranked long-term storage for all data object chunks. Thus, the amount of data object chunks stored on this storage is the biggest among all storages.

If we consider the used storage space of all storages in Figure 6.6b, we can observe that the two standard storages Amazon S3 Northern Virginia and Amazon S3 Northern California nearly use as much storage space as the long-term storage Amazon S3 Northern Virginia IA. Also notable is the fact that Amazon S3 Tokyo has the second most data object chunks stored, but its used storage space is nearly zero. From this observation we can substantiate the previously made statement that the Amazon S3 Tokyo storage is only used for small and medium sized data objects. Thus, the Amazon S3 Tokyo storage does not account for a big amount of the total incurred cost although it has the second highest amount of data objects stored.

Note that this approach does not migrate any data object chunks from standard storages to long-term storages. This is due to the definition of the latency consideration approach, as described in Section 5.4. Since we want to keep the latencies at the lowest possible level, we do not migrate data object chunks to long-term storages if the latency gets worse. Therefore, this approach only uses one long-term storage, i.e., the Amazon S3 IA storage which is used for all data objects that are uploaded.

Evaluation Results

Figure 6.7 shows the cumulative cost of both parts of the latency evaluation scenario, i.e., TU Wien and Tokyo, in comparison to the heuristic approach without latency consideration and the baseline.

It can be observed from the graph that the cost of both the heuristic approach and the latency consideration approaches increase right at the start of the evaluation. This is due to the already mentioned procedure, i.e., the heuristic approach and the latency consideration approach immediately store one of three data object chunks of every new data object that is uploaded on a long-term storage. Therefore, the cost graph rises due to the BTU cost that are immediately charged. After about 100 hours, cost of the latency consideration approach based on TU Wien increases slightly compared to the baseline since the approach migrates a big amount of unused data object chunks

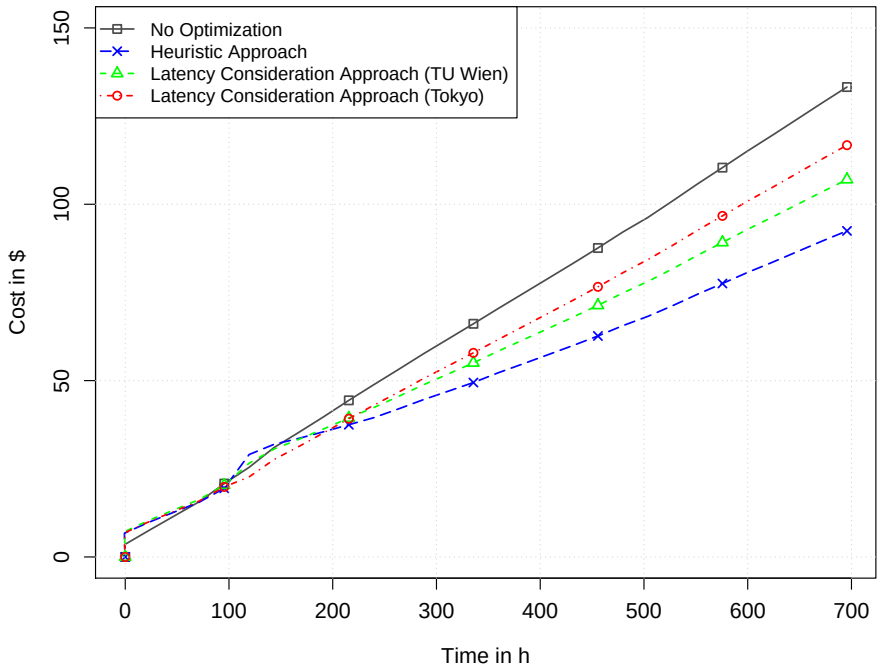


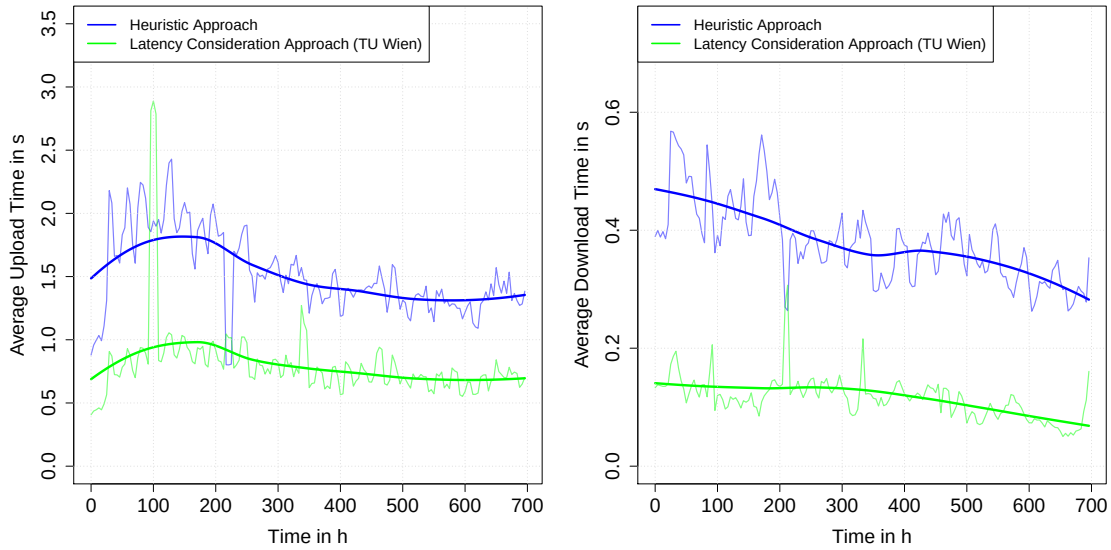
Figure 6.7: Total Cost of the different Optimization Approaches for the Latency Evaluation Scenario

from standard storages to long-term storages. After this migration, the graph flattens again and rises slower than the baseline. Since the latency consideration approach based on Tokyo does not migrate data object chunks to long-term storages, this approach is temporarily cheaper than the heuristic approach. However, after 200 hours the cost of the heuristic approach fall below the cost of the latency consideration approach. From there, the heuristic approach stays the cheapest approach among all shown approaches.

Note that both latency consideration approaches incur increased cost in comparison to the heuristic approach. This is due to the cost ranking of the latency consideration approaches which does not only aim at cost minimization but also incorporates latencies. That is, we accept a slightly increased cost on the one hand for yielding reduced latencies on the other hand. Since the latency consideration approach based on Tokyo does not migrate chunks to long-term storages, the cost of this approach are slightly higher than the cost of the latency approach based on TU Wien.

In summary, it can be stated that both latency consideration approaches yield slightly higher cost in comparison to the heuristic approach. However, both approaches still incur less cost than the baseline, which proves the correct functionality of our developed optimization approach.

In comparison to the baseline, we are able to save 12% with the latency consideration



(a) Upload Latencies

(b) Download Latencies

Figure 6.8: Latencies of the Latency Consideration Approach based on the Location of TU Wien

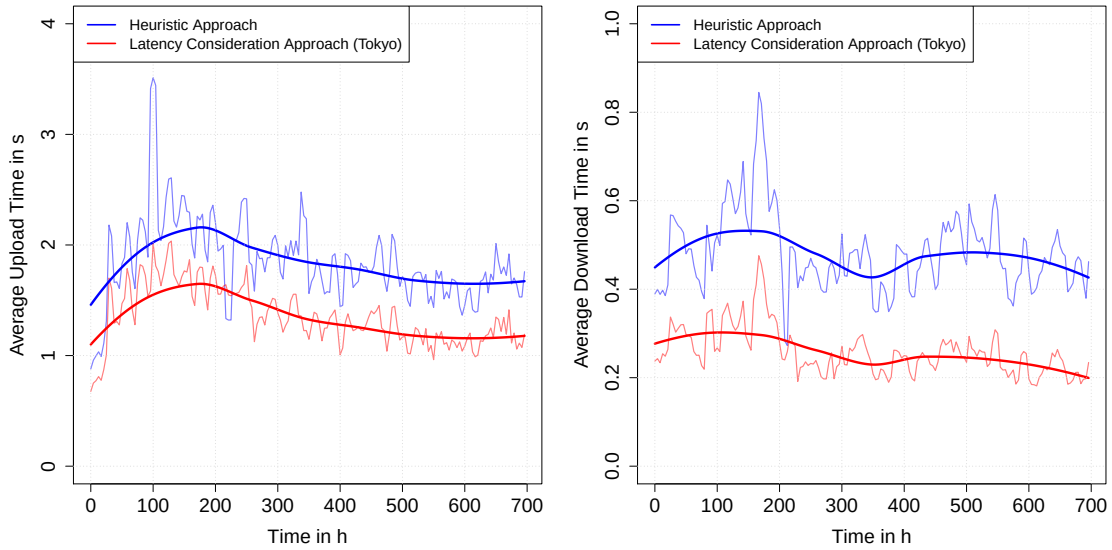
approach based on Tokyo and 20% with the latency consideration approach based on TU Wien. Compared to the heuristic, the latency consideration approach based on Tokyo incurs 19% more cost, whereas the latency consideration approach based on TU Wien incurs 11% more cost.

6.4.3 Latency Assessment

Figures 6.8 and 6.9 show the latencies for the latency consideration approaches based on TU Wien and Tokyo, respectively. The average upload times can be seen in Figures 6.8a and 6.9a, whereas the average download times are shown in Figures 6.8b and 6.9b.

We can observe from the graphs that both latency consideration approaches show reduced latencies for uploading and downloading data objects. From Figure 6.8a we can observe that the average upload time of the latency consideration approach based on TU Wien is 48% lower in comparison to the heuristic approach. Figure 6.8b shows that we yield a 69% lower download time with the latency consideration approach based on TU Wien if we compare it with the heuristic approach.

According to Figure 6.9a, the latency consideration approach based on Tokyo shows 26% less upload time compared to the heuristic approach. Finally, it can be seen from Figure 6.9b that the average download time is 46% less with the latency consideration approach based on Tokyo if it is compared to the heuristic approach.



(a) Upload Latencies

(b) Download Latencies

Figure 6.9: Latencies of the Latency Consideration Approach based on the Location of Tokyo

Table 6.2: Average Optimization Durations and Standard Deviations in ms

Days	Global Optimization Small Scale Scenario	Heuristic Approach Small Scale Scenario	Heuristic Approach Large Scale Scenario
1 - 5	37,672.89 (σ = 39,498.51)	1.51 (σ = 8.00)	4.24 (σ = 65.50)
6 - 10	54,551.53 (σ = 19,203.83)	5.32 (σ = 22.93)	8.20 (σ = 5.29)
11 - 15	48,848.48 (σ = 3,612.57)	0.07 (σ = 0.26)	8.08 (σ = 3.46)
16 - 20	55,073.10 (σ = 27,647.11)	0.02 (σ = 0.14)	8.38 (σ = 33.69)
21 - 25	65,689.16 (σ = 40,513.28)	0.05 (σ = 0.67)	8.72 (σ = 4.70)
26 - 30	62,538.28 (σ = 35,688.60)	0.02 (σ = 0.26)	13.55 (σ = 240.68)

6.5 Performance Assessment

In this section we evaluate the different optimization approaches from a performance perspective. We compare the global optimization with the heuristic approach by using the results of the small scale scenario as described in Section 6.2 as well as the large scale scenario as described in Section 6.3. We assess both optimization approaches by analyzing their average optimization durations and their used amount of metadata.

6.5.1 Optimization Duration

Table 6.2 shows the average optimization durations including the standard deviation for the global optimization and the heuristic approach over the whole duration of the

evaluation. It can be observed from the table that the average optimization duration for the global optimization is more than 54s in average for the small scale scenario. In comparison to this, the average optimization duration of the heuristic approach is only slightly over 1ms. Hence, the heuristic approach is about 50,000 times faster than the global optimization for the same scenario. This is due to the fact that the global optimization considers all possible combinations of data object chunks and storages in every optimization step. On the other hand, the heuristic approach does not build and solve an optimization problem but rather selects the placement by using the ranking function of each storage.

For the large scale scenario, the heuristic approach only needs 8.5ms in average. In summary, to determine a new cost-efficient placement, the heuristic approach only takes 8 times as long for the large scale scenario as for the small scale scenario. This underlines the exceptional scalability of the heuristic approach since the amount of data objects is more than 330 times bigger in the large scale scenario as opposed to the small scale scenario.

6.5.2 Metadata

Since all optimization approaches use historical information about past data object access, CORA needs to store this information in a database. Each optimization approach uses the history information of the last BTU in order to obtain a new cost-efficient placement solution. Therefore, CORA has to store the information of one complete BTU at every point in time. For the small scale scenario, the amount of metadata aggregates to 2.5MB. For the large scale scenario, the amount of metadata amounts to 865MB.

6.6 Summary of Results

In summary it can be said that we achieved good results for all evaluation scenarios. In the small scale scenario in Section 6.2 we analyzed the behavior of the global optimization and compared the results with the heuristic approach and the local optimization from [49]. We were able to reduce total cost by 37% with the global optimization and by 34% with the heuristic approach in comparison to the baseline.

In the large scale scenario in Section 6.3 we showed the scalability of the heuristic approach. We evaluated this approach with a total of 100,000 data objects. In this scenario we showed a reduction of total cost by 25% compared to the baseline.

In Section 6.4 we evaluated the latency consideration approach based on two different locations in order to show the different behavior of the optimization depending on the location of the middleware. On the one hand, we evaluated the approach based on our current location at TU Wien. On the other hand, we evaluated the approach based on the location of the Amazon EC2 region Tokyo. For simulating the location of the middleware we performed several measurements upfront. We used these measurements to obtain the latencies of the various used cloud storages. By incorporating latencies

into the selection of the best fitting storages, we showed that we are able to achieve greatly reduced latencies with this approach. In particular, the average upload times were reduced by 26% for Tokyo and by 48% for TU Wien. Additionally, the average download times could be reduced by 46% for Tokyo and by 69% for TU Wien. The latency consideration approach based on the location of Tokyo yielded in a cost reduction of 12%. With the latency consideration approach based on the location of TU Wien we were able to save cost in the amount of 20%.

Finally, in Section 6.5 we evaluated the global optimization and the heuristic from a performance perspective. We analyzed the average optimization durations of both approaches and compared the used amount of metadata that the middleware stores in the small scale and large scale evaluation scenario. By comparing the average optimization durations, we further outlined the scalability potential of the heuristic approach over the global optimization.

CHAPTER 7

Conclusion and Future Work

"Don't watch the clock; do what it does. Keep going."

Sam Levenson

In this last chapter we will conclude the thesis by summarizing the main parts of this work. First, we will outline the contributions of our work and, subsequently, we will describe possibilities for extending our work in the future.

7.1 Contributions

Cloud storages have become a popular alternative to traditional local storage systems in the past few years. Compared to local storage systems, cloud storages offer a lot of advantages to their users. Some of the advantages are better data integrity, higher data availability and extended durability as well as lower administration cost. Therefore, users of cloud storage services can benefit a lot from these properties. However, vendor lock-in is one of the biggest issues when using a particular cloud storage service. If customers rely on only one single cloud storage provider, the possibility of service unavailability exists. The provider could even go out of business which could lead to a loss of all stored data. A recent service outage of Amazon S3 showed that also big cloud storage providers suffer from service unavailability [5].

To overcome the problem of vendor lock-in, we designed an approach that combines multiple cloud storages to store the data in a redundant manner. We focused on optimizing the placement of data objects among multiple cloud storages. The main goal of the optimization was to provide a cost-efficient placement solution. To achieve this goal, we designed three different optimization approaches. The first approach was the formulation

of a global optimization problem. This problem always provides a cost-efficient placement solution since the placement of all data objects is optimized every time a data object access takes place. With the second optimization approach, we designed a heuristic model with the aim of better runtime performance and scalability in comparison to the global optimization, while still providing an acceptable placement solution in terms of total cost. The heuristic approach does not always select the optimal set of cloud storages. Still, it comes very close to the global optimization in terms of cost efficiency. The third approach was an approach that incorporates latencies of the cloud storages into the selection of the best-fitting set of cloud storages. By extending the heuristic approach, we ensured the good scalability properties of the heuristic model. We designed the latency consideration approach by taking into consideration the latencies of different storage regions that were measured upfront. By accepting a slight increase of total cost, we could instead ensure the lowest possible latencies between the middleware and the cloud storages. This leads to the benefit of faster data access which provides a better end-user experience.

In the first part of the thesis we identified the research challenge by analyzing related work in the field of cloud-based storage middlewares. The research challenge was to design an optimization algorithm that aims at cost minimization while considering Block Rate Pricing models of the different cloud storage providers and data access patterns to improve the data object placement. Moreover, we used erasure coding as redundancy mechanism and also used long-term cloud storages in addition to standard storages. We tackled this research challenge by formulating a global optimization problem as a MILP model. Furthermore, we designed a heuristic approach and a latency consideration approach as alternatives to the global optimization. By designing and implementing these three different approaches in one work, we filled the research gap since, to the best of our knowledge, there exists no work that covers all of the analyzed features.

After the design and implementation of all three optimization approaches we evaluated each approach in detail. For that reason, we introduced three different evaluation scenarios. In the first part we proved the correct functionality of the global optimization in a small scale scenario. To verify the scalability property of the heuristic approach we evaluated it in a large scale scenario with a big amount of data objects. Finally, we analyzed the functionality of the latency consideration approach in a scenario where we also outlined the benefits of this approach by comparing the resulting latencies to the heuristic approach without latency consideration. We did this by evaluating the latency consideration approach based on two different locations. Based on the location we explained the functionality and showed the different behavior of the storage selection based on the location of the middleware. For all three optimization approaches we showed essential cost savings by comparing the results of each approach to a baseline.

7.2 Future Work

This thesis provides a solid design and description of the functionality of three different optimization approaches with the aim of cost and latency minimization. Therefore, this work can be used as foundation for further research projects in the area of cloud-based storage solutions that aim at minimizing overall storage cost.

Global Optimization

The optimization model could be enhanced by a more accurate prediction of future data object usage. We observed from our detailed evaluations that the majority of data objects from the used file access trace are rarely used data objects. Therefore, the optimization could follow a more optimistic approach by sooner storing data object chunks on long-term storages. Moreover, predicting future data object usage could be extended by taking into account the size and type of data objects. For instance, a big data object that is packed in an archive format is most likely a backup archive and could therefore be directly stored on a long-term storage which could yield even higher cost savings.

Heuristic Approach

We designed the heuristic approach to migrate unused data object chunks from standard storages to long-term storages after a predefined time interval. However, we did not include the functionality to transfer frequently used data object chunks from long-term storages back to standard storages. Implementing this functionality would improve the cost savings of the heuristic approach even more since frequently accessed data object chunks may incur high cost due to the higher outgoing data transfer cost and the data retrieval cost that are usually not charged for chunks on standard storages.

Latency Consideration Approach

Since we measured the latencies of different storage regions upfront, we used a static set of measurement data in the optimization. The latency consideration approach could be extended by incorporating real-time latency monitoring. This would yield even more accurate placement solutions by reducing the latencies between the middleware and the used cloud storages.

CORA

The used middleware CORA is currently implemented in a centralized architecture. To further extend the performance and scalability of the whole system, CORA could be redesigned to follow a decentralized architecture.

APPENDIX A

Variables of the Global Optimization Problem

Table A.1: Variables of the Global Optimization Problem

Variable	Description
$s \in S = \{s_1, s_2, \ldots\}$	S is the set of available cloud storages and s is a specific cloud storage.
$f \in F = \{f_1, f_2, \ldots\}$	F is the set of all data objects and f indicates a specific data object.
$k \in K_f = \{k_{f1}, k_{f2}, \ldots\}$	K_f is the set of data object chunks of data object f and k indicates a specific data object chunk.
n	n is the amount of data object chunks. Each data object is split into n chunks, i.e., $\lvert K_f \rvert = n$.
m	m is the amount of data object chunks that is needed to successfully reconstruct a data object.
$b_s \in B_s^{st} = \{b_{s1}, b_{s2}, \ldots\}$	B_s^{st} is the set of all price steps of the storage cost for storage s and b_s indicates one specific price step.
$b_s \in B_s^{T_{out}} = \{b_{s1}, b_{s2}, \ldots\}$	$B_s^{T_{out}}$ is the set of all price steps of the outgoing data transfer cost for storage s and b_s indicates one specific price step.
$b_s = (b_s^L, b_s^U, p_s)$	b_s^L is the lower bound of price step b_s, b_s^U is the upper bound of price step b_s and p_s is the actual price for this step.
Continued on next Page	

Variable	Description
$c_{(s,k,\tau)}$	$c_{(s,k,\tau)}$ are the total cost for storing a data object chunk k on cloud storage s by taking the usage history of the last τ minutes into account.
$c^{S}_{(s,k,\tau)}$	$c^{S}_{(s,k,\tau)}$ are the storage cost.
$c^{R}_{(s,k,\tau)}$	$c^{R}_{(s,k,\tau)}$ are the read request cost.
$c^{W}_{(s,k,\tau)}$	$c^{W}_{(s,k,\tau)}$ are the write request cost.
$c^{T_{in}}_{(s,k,\tau)}$	$c^{T_{in}}_{(s,k,\tau)}$ are the incoming data transfer cost.
$c^{T_{out}}_{(s,k,\tau)}$	$c^{T_{out}}_{(s,k,\tau)}$ are the outgoing data transfer cost.
$c^{M}_{(s_1,s_2,k)}$	$c^{M}_{(s_1,s_2,k)}$ is the migration cost for transferring chunk k from s_1 to s_2 if the providers of s_1 and s_2 are different.
$c^{M_{red}}_{(s_1,s_2,k)}$	$c^{M_{red}}_{(s_1,s_2,k)}$ is the migration cost for transferring chunk k from s_1 to s_2 if the provider of s_1 and s_2 is the same.
$p^{S}_{(s,\gamma_{(s,k)})}$	$p^{S}_{(s,\gamma_{(s,k)})}$ calculates the storage price for storing data object chunk k on cloud storage s.
p^{R}_{s}	p^{R}_{s} defines the price per read operation.
p^{W}_{s}	p^{W}_{s} defines the price per write operation.
$p^{T_{in}}_{(s,\beta_{(s,k)})}$	$p^{T_{in}}_{(s,\beta_{(s,k)})}$ is the incoming data transfer price for the cloud storage s.
$p^{T_{out}}_{(s,\beta_{(s,k)})}$	$p^{T_{out}}_{(s,\beta_{(s,k)})}$ is the outgoing data transfer price for the cloud storage s.
p^{ret}_{s}	p^{ret}_{s} is the data retrieval price.
$r^{R}_{(k,\tau)}$	$r^{R}_{(k,\tau)}$ expresses the number of read requests that were performed on data object chunk k in the last τ minutes.
$r^{W}_{(k,\tau)}$	$r^{W}_{(k,\tau)}$ expresses the number of write requests that were performed on data object chunk k in the last τ minutes.
$t^{in}_{(k,\tau)}$	$t^{in}_{(k,\tau)}$ defines the amount of bytes that were written to the cloud storage during time period τ.
$t^{out}_{(k,\tau)}$	$t^{out}_{(k,\tau)}$ defines the amount of bytes that were read from the cloud storage during time period τ.

Continued on next Page

Variable	Description
$\beta_{(s,k)}$	$\beta_{(s,k)}$ calculates the amount of transferred data of cloud storage s in the current billing period.
$\gamma_{(s,k)}$	$\gamma_{(s,k)}$ calculates the used storage space of data object chunk k on cloud storage s in the current billing period.
$\sigma_{(k,\tau)}$	$\sigma_{(k,\tau)}$ defines the size of data object chunk k in the last τ minutes.
$\hat{\sigma}_{(k,BTU)}$	$\hat{\sigma}_{(k,BTU)}$ is the size of data object chunk k, which is charged for the remaining time until the BTU is reached.

Table A.2: Decision Variables of the Global Optimization Problem

Decision Variable	Description
$x_{(s,k)} \in \{0,1\}$	$x_{(s,k)} \in \{0,1\}$ marks if a data object chunk k is stored on cloud storage s. If k is stored on s, then $x_{(s,k)} = 1$, otherwise $x_{(s,k)} = 0$.
$h_k \in \{0,1\}$	h_k defines if data object chunk k is currently stored on a long-term storage ($h_k = 1$) or on a standard storage ($h_k = 0$).
$\hat{h}_s \in \{0,1\}$	$\hat{h}_s$ states if storage s is a long-term storage ($\hat{h}_s = 1$) or a standard storage ($\hat{h}_s = 0$).
$z_{(s_1,s_2)} \in \{0,1\}$	$z_{(s_1,s_2)}$ states if two storages s_1 and s_2 have different storage providers. If s_1 and s_2 have different providers, then $z_{(s_1,s_2)} = 1$, otherwise (i.e., if they have the same provider) $z_{(s_1,s_2)} = 0$.
$y_{(s_1,s_2)} \in \{0,1\}$	$y_{(s_1,s_2)}$ states if two storages s_1 and s_2 have the same storage provider. If s_1 and s_2 have the same provider, then $y_{(s_1,s_2)} = 1$, otherwise (i.e., if they have different providers) $y_{(s_1,s_2)} = 0$.
$g_{(\tilde{S},f)} \in \{0,1\}$	$g_{(\tilde{S},f)}$ states if every cloud storage $s \in \tilde{S}$ has a data object chunk $k \in K_f$ stored. If this is the case, then $g_{(\tilde{S},f)} = 1$, otherwise (i.e., if at least one of the storages does not have a data object chunk of K_f stored) $g_{(\tilde{S},f)} = 0$.
$u^{st}_{(s,b_s)} \in \{0,1\}$	If the used storage space of storage s is greater than the lower bound b^L_s, then $u^{st}_{(s,b_s)} = 1$, otherwise $u^{st}_{(s,b_s)} = 0$.
$v^{st}_{(s,b_s)} \in \{0,1\}$	If the used storage space of storage s is lower than the upper bound b^U_s, then $v^{st}_{(s,b_s)} = 1$, otherwise $v^{st}_{(s,b_s)} = 0$.
$o^{st}_{(s,b_s)} \in \{0,1\}$	$o^{st}_{(s,b_s)} = 1$ indicates that the used storage space is between the lower and the upper bound of b_s, otherwise $o^{st}_{(s,b_s)} = 0$.
$u^{T_{out}}_{(s,b_s)} \in \{0,1\}$	If the outgoing data transfer of storage s is greater than the lower bound b^L_s, then $u^{T_{out}}_{(s,b_s)} = 1$, otherwise $u^{T_{out}}_{(s,b_s)} = 0$.
$v^{T_{out}}_{(s,b_s)} \in \{0,1\}$	If the outgoing data transfer of storage s is lower than the upper bound b^U_s, then $v^{T_{out}}_{(s,b_s)} = 1$, otherwise $v^{T_{out}}_{(s,b_s)} = 0$.
$o^{T_{out}}_{(s,b_s)} \in \{0,1\}$	$o^{T_{out}}_{(s,b_s)} = 1$ indicates that the outgoing data transfer is between the lower and the upper bound of b_s, otherwise $o^{T_{out}}_{(s,b_s)} = 0$.

APPENDIX B

Latency Measurements for Tokyo

Table B.1: Average Upload Times and Standard Deviations based on the Location of Tokyo in s

File Size	Frankfurt	N. Virginia	N. California	São Paulo	Tokyo
1KB	0.6 ($\sigma = 0.4$)	0.4 ($\sigma = 0.2$)	0.4 ($\sigma = 0.2$)	0.7 ($\sigma = 0.4$)	0.07 ($\sigma = 0.05$)
10 KB	0.5 ($\sigma = 0.02$)	0.3 ($\sigma = 0.03$)	0.3 ($\sigma = 0.05$)	0.6 ($\sigma = 0.03$)	0.04 ($\sigma = 0.01$)
100 KB	1.2 ($\sigma = 0.09$)	0.8 ($\sigma = 0.06$)	0.6 ($\sigma = 0.02$)	1.4 ($\sigma = 0.04$)	0.06 ($\sigma = 0.02$)
1 MB	2.5 ($\sigma = 0.09$)	1.8 ($\sigma = 0.3$)	1.7 ($\sigma = 0.2$)	3.8 ($\sigma = 0.3$)	0.2 ($\sigma = 0.2$)
10 MB	4.0 ($\sigma = 0.1$)	3.5 ($\sigma = 0.2$)	2.7 ($\sigma = 0.04$)	5.8 ($\sigma = 0.2$)	0.5 ($\sigma = 0.2$)
100 MB	14.0 ($\sigma = 0.4$)	10.7 ($\sigma = 0.7$)	9.2 ($\sigma = 1.4$)	17.1 ($\sigma = 0.5$)	3.1 ($\sigma = 0.2$)

Table B.2: Average Download Times and Standard Deviations based on the Location of Tokyo in s

File Size	Frankfurt	N. Virginia	N. California	São Paulo	Tokyo
1 KB	0.3 ($\sigma = 0.02$)	0.2 ($\sigma = 0.03$)	0.1 ($\sigma = 0.01$)	0.3 ($\sigma = 0.008$)	0.02 ($\sigma = 0.01$)
10 KB	0.3 ($\sigma = 0.03$)	0.2 ($\sigma = 0.02$)	0.1 ($\sigma = 0.001$)	0.3 ($\sigma = 0.02$)	0.02 ($\sigma = 0.004$)
100 KB	0.3 ($\sigma = 0.04$)	0.2 ($\sigma = 0.05$)	0.1 ($\sigma = 0.005$)	0.3 ($\sigma = 0.01$)	0.03 ($\sigma = 0.01$)
1 MB	1.3 ($\sigma = 0.06$)	0.8 ($\sigma = 0.04$)	0.6 ($\sigma = 0.2$)	0.3 ($\sigma = 0.009$)	0.09 ($\sigma = 0.03$)
10 MB	1.3 ($\sigma = 0.2$)	0.9 ($\sigma = 0.2$)	0.7 ($\sigma = 0.04$)	0.3 ($\sigma = 0.008$)	0.09 ($\sigma = 0.02$)
100 MB	1.4 ($\sigma = 0.07$)	0.9 ($\sigma = 0.05$)	0.7 ($\sigma = 0.05$)	0.3 ($\sigma = 0.02$)	0.08 ($\sigma = 0.004$)

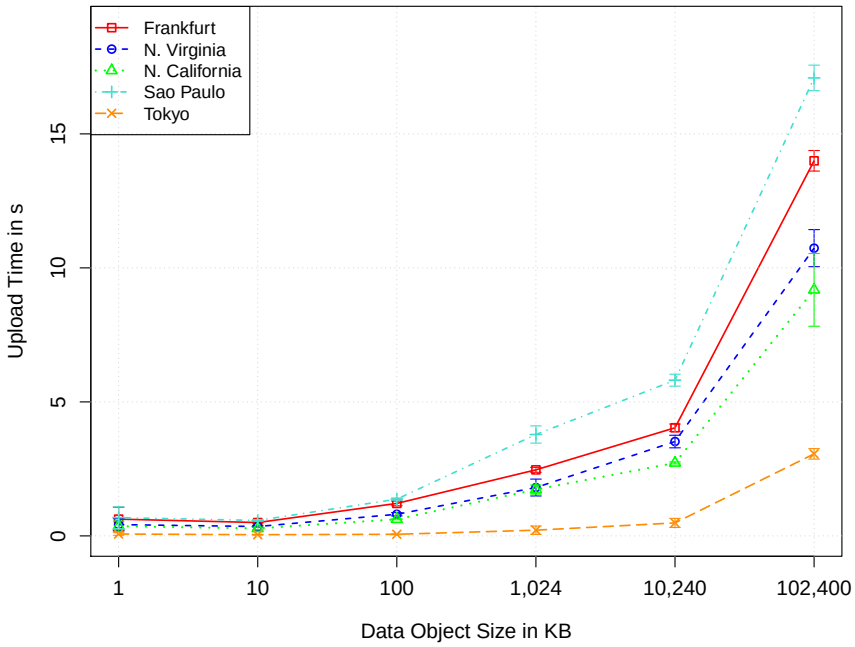


Figure B.1: Comparison of the Upload Times of Different Storage Regions based on the Location of Tokyo

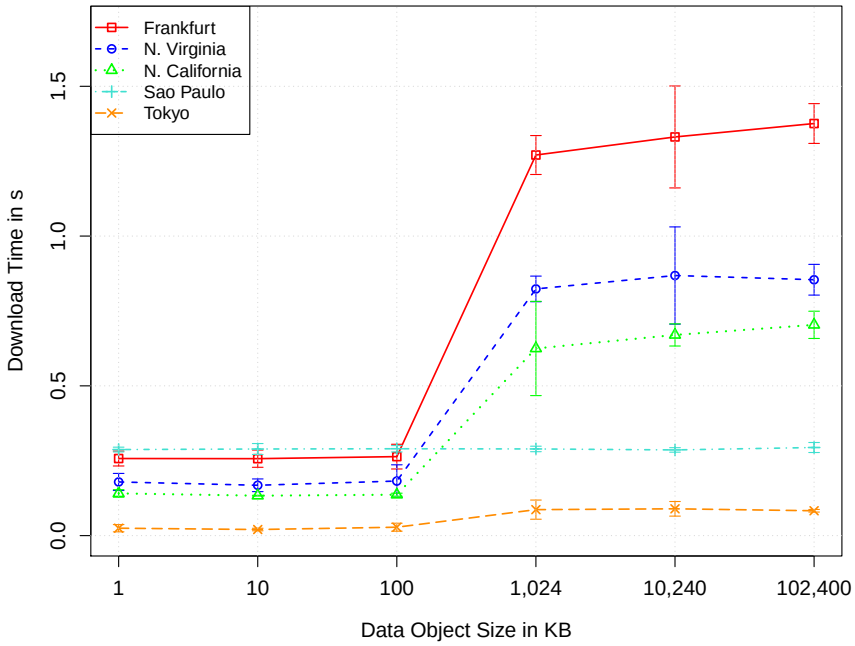


Figure B.2: Comparison of the Download Times of Different Storage Regions based on the Location of Tokyo

List of Figures

List of Tables

List of Algorithms

Acronyms

API Application Programming Interface. 7, 8, 29, 49

BSU Billing Storage Unit. 9, 31, 32, 61

BTU Billing Time Unit. 9, 31, 32, 59, 63–65, 70, 74, 83

FTP File Transfer Protocol. 7

GAP Generalized Assignment Problem. 40, 41

IA Infrequent Access. 9, 58, 59, 61, 65, 68–70

ILP Integer Linear Programming. 13

KP Knapsack Problem. 12, 24

MILP Mixed Integer Linear Programming. 13, 20, 22, 25, 78

MINGAP Minimization Version of the Generalized Assignment Problem. 41

MKP Multidimensional Knapsack Problem. 12, 13, 20, 40

NIST National Institute of Standards and Technology. 6

QoS Quality of Service. 1–3, 5, 16, 17, 19, 35, 38, 39

RAID Redundant Array of Independent Disks. 15, 21, 23

RTT Round-Trip Time. 17

SLO Service Level Objective. 17, 30, 38, 59

SPoF Single Point of Failure. 15

StaaS Storage as a Service. 7

TSP Traveling Salesperson Problem. 11, 12

Bibliography

[1] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A Case for Cloud Storage Diversity. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*, pages 229–240, 2010.

[2] M. Aguilera, R. Janakiraman, and Lihao Xu. Using Erasure Codes Efficiently for Storage in a Distributed System. In *2005 International Conference on Dependable Systems and Networks (DSN)*, pages 336–345, 2005.

[3] M. Alhamad, T. Dillon, and E. Chang. Conceptual SLA framework for cloud computing. In *4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, pages 606–610, 2010.

[4] E. Allen and C. M. Morris. Library of Congress and DuraCloud Launch Pilot Program Using Cloud Technologies to Test Perpetual Access to Digital Content. The Library of Congress, News Release, 2009. URL `https://www.loc.gov/today/pr/2009/09-140.html`. Accessed: October 2016.

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

[6] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[7] J. Barr. AWS Storage Update – S3 & Glacier Price Reductions + Additional Retrieval Options for Glacier, 2016. URL `https://aws.amazon.com/blogs/aws/aws-storage-update-s3-glacier-price-reductions/`. Accessed: November 2016.

[8] A. Bergen, Y. Coady, and R. McGeer. Client bandwidth: The forgotten metric of online storage providers. In *2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*, pages 543–548, 2011.

[9] D. Bermbach, M. Klems, S. Tai, and M. Menzel. MetaStorage: A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs. In *2011 IEEE 4th International Conference on Cloud Computing (CLOUD)*, pages 452–459, 2011.

[10] D. Bermbach, T. Kurze, and S. Tai. Cloud Federation: Effects of Federated Compute Resources on Quality of Service and Cost. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 31–37, 2013.

[11] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and Secure Storage in a Cloud-of-Clouds. *ACM Transactions on Storage*, 9(4): 12:1–12:33, 2013.

[12] K. D. Bowers, A. Juels, and A. Oprea. HAIL: A High-availability and Integrity Layer for Cloud Storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pages 187–198, 2009.

[13] B. Butler. Gartner: Top 10 cloud storage providers. Network World, 2013. URL `http://www.networkworld.com/article/2162466/cloud-computing/cloud-computing-gartner-top-10-cloud-storage-providers.html`. Accessed: November 2016.

[14] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[15] C.-W. Chang, P. Liu, and J.-J. Wu. Probability-Based Cloud Storage Providers Selection Algorithms with Maximum Availability. In *2012 41st International Conference on Parallel Processing (ICPP)*, pages 199–208, 2012.

[16] Y.-F. R. Chen. The Growing Pains of Cloud Storage. *IEEE Internet Computing*, 19 (1):4–7, 2015.

[17] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras. Inside dropbox. In *Proceedings of the 2012 Internet Measurement Conference (IMC '12)*, pages 481–494, 2012.

[18] A. Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

[19] A. Fréville and G. Plateau. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics*, 49(1):189–212, 1994.

[20] S. Garfinkel. An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS. Technical Report TR-08-07, Computer Science Group, Harvard University, Cambridge, 2007.

[21] 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 *Proceedings of the 2015 Internet Measurement Conference (IMC '15)*, pages 155–168, 2015.

[22] P. Gupta, A. Seetharaman, and J. R. Raj. The usage and adoption of cloud computing by small and medium businesses. *International Journal of Information Management*, 33(5):861–874, 2013.

[23] G. Gutin and A. P. Punnen. *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*. Springer US, 2007.

[24] V. Kann. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Royal Institute of Technology, Stockholm, 1992.

[25] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Berlin Heidelberg, 2004.

[26] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University and Department of Computer Science, University of Durham, 2007.

[27] O. E. Kundakcioglu and S. Alizamir. Generalized Assignment Problem. In *Encyclopedia of Optimization*, pages 1153–1162. Springer US, 2008.

[28] D. Kusnetzky. Nasuni Cloud Storage Gateway, 2010. URL `http://www.zdnet.com/article/nasuni-cloud-storage-gateway/`. Accessed: November 2016.

[29] S. Liu, X. Huang, H. Fu, and G. Yang. Understanding Data Characteristics and Access Patterns in a Cloud Storage System. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 327–334, 2013.

[30] Y. Ma, T. Nandagopal, K. P. N. Puttaswamy, and S. Banerjee. An ensemble of replication and erasure codes for cloud file systems. In *Proceedings of the IEEE INFOCOM 2013*, pages 1276–1284, 2013.

[31] A. Mahanti, D. Eager, and C. Williamson. Temporal locality and its impact on Web proxy cache performance. *Performance Evaluation*, 42(2):187–203, 2000.

[32] Y. Mansouri, A. N. Toosi, and R. Buyya. Brokering Algorithms for Optimizing the Availability and Cost of Cloud Storage Services. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 581–589, 2013.

[33] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[34] J. McKendrick. Cloud Computing's Vendor Lock-In Problem: Why the Industry is Taking a Step Backward. Forbes, 2011. URL `http://www.forbes.com/sites/joemckendrick/2011/11/20/cloud-computings-vendor-lock-in-problem-why-the-industry-is-taking-a-step-backwards/`. Accessed: October 2016.

[35] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical Report SP 800-145, National Institute of Standards and Technology, Gaithersburg, 2011.

[36] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Comprehensive QoS monitoring of Web services and event-based SLA violation detection. In *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing (MWSOC '09)*, pages 1–6, 2009.

[37] M. Naldi and L. Mastroeni. Cloud Storage Pricing: A Comparison of Current Practices. In *Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services (HotTopiCS '13)*, pages 27–34, 2013.

[38] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science grids. In *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing (DADC '08)*, pages 55–64, 2008.

[39] T. G. Papaioannou, N. Bonvin, and K. Aberer. Scalia: An Adaptive Scheme for Efficient Multi-cloud Storage. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*, pages 20:1–20:10, 2012.

[40] D. A. Patterson, G. Gibson, R. H. Katz, D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data (SIGMOD '88)*, pages 109–116, 1988.

[41] 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 *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[42] W. W. Peterson and J. E. J. Weldon. *Error-correcting codes*. The MIT Press, 1972.

[43] C. Pettey. Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017. Gartner, Press Release, 2017. URL `http://www.gartner.com/newsroom/id/3616417`. Accessed: April 2017.

[44] J. S. Plank. Erasure Codes for Storage Systems: A Brief Primer. *;login: The Usenix Magazine*, 38(6):44–50, 2013.

[45] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[46] R. Rodrigues and B. Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *Peer-to-Peer Systems IV*, pages 226–239. Springer Berlin Heidelberg, 2005.

[47] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, and S. Dustdar. Winds of change: From vendor lock-in to the meta cloud. *IEEE Internet Computing*, 17(1):69–73, 2013.

[48] M. Schnjakin, T. Metzke, and C. Meinel. Applying Erasure Codes for Fault Tolerance in Cloud-RAID. In *2013 IEEE 16th International Conference on Computational Science and Engineering (CSE)*, pages 66–75, 2013.

[49] P. Waibel, C. Hochreiner, and S. Schulte. Cost-Efficient Data Redundancy in the Cloud. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–9, 2016.

[50] H. Weatherspoon and J. D. Kubiatowicz. Erasure Coding Vs. Replication: A Quantitative Comparison. In *Peer-to-Peer Systems*, pages 328–337. Springer Berlin Heidelberg, 2002.

[51] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. SPANStore. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*, pages 292–308, 2013.

[52] W. Zeng, Y. Zhao, K. Ou, and W. Song. Research on Cloud Storage Architecture and Key Technologies. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (ICIS '09)*, pages 1044–1048, 2009.

[53] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

[54] 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 Transactions on Cloud Computing*, 3(3):372–386, 2015.