Dynamic data replication for distributed cloud storage

Cai, Jianfei

2021

Cai, J. (2021). Dynamic data replication for distributed cloud storage. Master's thesis, Nanyang Technological University, Singapore. <https://hdl.handle.net/10356/149825>

<https://hdl.handle.net/10356/149825>

<https://doi.org/10.32657/10356/149825>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 10:25:24 SGT

DYNAMIC DATA REPLICATION FOR DISTRIBUTED CLOUD STORAGE

Cai Jianfei

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Master of Engineering (M.Eng)

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

12/15/2020

.....

Date

Cai Jianfei

.....

Cai Jianfei

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

12/15/2020

.....
Date

.....
Tang Xueyan

Authorship Attribution Statement

This thesis **does not** contain any materials from papers published in peer-reviewed journals or from papers accepted at conferences in which I am listed as an author.

12/15/2020

.....

Date

Cai Jianfei

.....

Cai Jianfei

Acknowledgements

I would like to express my sincere thanks and appreciation to my supervisor, A/P Tang Xueyan, for his guidance and patience in imparting valuable knowledge to me. We had many discussions during my candidature as a M.Eng Student, he always encouraged me and gave me a lot of inspiration.

Besides, I want to thanks my parents and Wang Xixuan for their strong support during my M.Eng program.

Dynamic Data Replication for Distributed Cloud Storage

Cai Jianfei

Supervisor: Assoc. Prof. Tang Xueyan

Abstract

Cloud storage services have become increasingly popular in recent years. This thesis investigates a cost optimization problem for data replication in distributed cloud storage. We consider two different settings: offline and online. In the offline setting where the complete information of data access requests is available, we develop an $O(\max\{m^2n, n^2\})$ optimal algorithm, where m and n are the numbers of storage sites and requests respectively. In the online setting where the requests arriving in the future are not known, we propose two distributed algorithms: a 3-competitive basic algorithm and a dynamic algorithm. Both algorithms are lightweight and easy to implement. Experiments using the pricing data of Google Cloud Storage show that our online algorithms can perform close to the optimal.

List of Figures

3.1	An example of space-time diagram	9
4.1	Four types of requests	14
4.2	A replica at site s_x crossing time $t[r_j]$	17
4.3	r_g and $r_{p(g)}$	18
4.4	Recurrence for $D(i)$	20
4.5	$E(k)$ and $E'(k)$	22
4.6	A replica crosses time $t[r_j]$	24
4.7	Recurrence for $F(i)$	25
4.8	$s_x \neq s_{\min}$ for $G(i)$	28
5.1	Numerical results of cost saving	34
6.1	Cost for ASCEND setting	44
6.2	Cost for DESCEND setting	44
6.3	Cost for FLUCTUATE setting	45
6.4	Cost ratio between basic online and optimal offline algorithms	45
6.5	Cost ratio between dynamic online and optimal offline algorithms	46
6.6	Running time for optimal offline algorithm	48
6.7	Running time for dynamic online algorithm	49

List of Tables

3.1	Summary of notations	8
6.1	Google cloud storage pricing [1]	42

Contents

1	Introduction	1
2	Related Work	4
3	System Model and Definitions	7
4	An Optimal Offline Algorithm	10
4.1	Format of Optimal Replication Strategy	10
4.2	A Dynamic Programming Algorithm	15
4.2.1	Preliminaries	16
4.2.2	Recurrence for $D(i)$	20
4.2.3	Recurrence for $E(i)$	23
4.2.4	Recurrence for $F(i)$	25
4.2.5	Recurrence for $G(i)$	27
5	Online Algorithms	30
5.1	A Basic Online Algorithm	30
5.2	A Dynamic Online Algorithm	33
6	Experimental Evaluation	42
6.1	Experimental Setup	42
6.2	Experimental Results	43

Chapter 1

Introduction

With the explosive growth of data and rapid development of cloud computing technologies, deploying cloud storage services has become a widely used method to store massive data, maintain high availability and save cost. Different from traditional storage, clients do not need to possess and keep their own dedicated servers to satisfy increasing storage requirements. In contrast, they just need to buy storage services from cloud storage providers, adjust the amount of resource usage flexibly according to their demands, and pay for the actual amount of resources used under “pay-as-you-go” billing.

Data replication is an important technique to facilitate data access in distributed systems [2, 3]. In cloud storage services, replicas of a data object can be created and stored at sites dispersed at different geographical locations. Most cloud storage services, such as Amazon S3, Microsoft Azure and Google Cloud, charge clients according to the time duration and size of storage as well as the traffic going into and out of the storage sites. The prices normally vary across the sites. Such a billing model yields a trade-off between the storage and transfer costs. Creating more replicas of an object can reduce the transfer cost among various sites to access the object, but

it increases the storage cost. Vice versa, maintaining fewer replicas of an object can save the storage cost, but it induces the transfer cost when the object needs to be accessed at sites with no replicas. An optimal replication strategy should strike a balance between the storage and transfer costs.

In this thesis, we study a cost optimization problem for distributed cloud storage. Our goal is to dynamically create and delete the replicas of a data object to serve access requests in a cost-effective manner. We consider both the storage cost of the replicas and the transfer cost to create new replicas or serve requests. The storage costs at different sites are distinct, while the transfer costs between sites are assumed to be uniform (for example, in Google Cloud Storage, the transfer costs between any sites in the same continent are identical [1]).

Our contributions are summarized as follows:

(1) In the offline setting, given the full information of a request sequence, we develop a dynamic programming algorithm to compute the optimal set of replicas to create and hold over time that minimizes the total cost to serve all the requests. The algorithm has a time complexity of $O(\max\{m^2n, n^2\})$, where m and n are the numbers of sites and requests respectively.

(2) In the online setting, we propose a basic algorithm that has a strong worst-case performance guarantee, and enhance it to dynamically determine where to place the replicas based on the requests received and the cost estimations. Both algorithms are lightweight and distributed in nature.

(3) We conduct extensive simulations using the real pricing data of Google Cloud Storage and identify the factors that affect the relative performance of the algorithms.

The rest of this thesis is organized as follows: Chapter 2 summarizes the

related work. Chapter 3 describes the system model. Chapter 4 presents the optimal offline algorithm. The basic and enhanced online algorithms are proposed and analyzed in Chapter 5. Chapter 6 discusses the experimental setup and results. Finally, Chapter 7 concludes the thesis.

Chapter 2

Related Work

Cost optimization of cloud storage has attracted much attention in recent years. Agarwala *et al.* [4] developed a service called iCostale, which interfaces with multiple cloud storage providers. It reduces the data storage cost by careful selection and placement of data into one of many storage clouds, and transforms data based on the data type, access frequency, transformation overhead, and the cost model of the cloud storage providers. Bellur *et al.* [5] presented a solution for mapping applications onto cloud providers that offer distinct dynamic pricing schemes. The goal is to optimize the cost of a multi-site deployment consisting of compute and storage components at each site while satisfying service-level agreement requirements of the application.

In cloud storage systems, data replication is an important means to increase the data availability. Wei *et al.* [6] proposed a dynamic replication management scheme that computes and keeps minimal replica number for a given availability requirement. By adjusting the replica number based on workload changes and node capacity, it can dynamically redistribute workloads among data nodes in the heterogeneous cloud. Similarly, Gill *et al.* [7] presented an optimized data replication strategy that identifies the minimum

number of replicas required to guarantee the desired availability. They use the concept of knapsack to optimize the cost of replication and to migrate data from higher-cost data centers to lower-cost data centers, without compromising the data availability. Thanasis *et al.* [8] introduced Scalia, a cloud storage brokerage solution that continuously adapts data placement according to its access pattern and the optimization objectives. Scalia considers repositioning selected data objects that can significantly lower the storage cost. It mediates data placement across multiple public cloud providers and private cloud resources in order to help the data owners avoid vendor lock-in and meet availability and durability constraints in a cost-effective manner.

Wu *et al.* [9] presented SPANStore, a key-value store service that spans multiple cloud providers, estimates application workload and minimizes the use of compute resources to implement tasks to reduce cost. Long *et al.* [10] proposed an algorithm to seek near optimal solutions that balance the trade-offs among five optimization objectives including mean access latency, mean file unavailability, mean service time, load variance and energy consumption. Mansouri *et al.* [11] proposed data placement algorithms in the tiered cloud storage services that offer two storage tiers characterized by differentiated Quality of Service and different storage and access costs. The hot tier has higher storage cost but lower access cost, which makes it more suitable for storing frequently accessed data objects. In contrast, the cool tier has lower storage cost but higher access cost, which makes it more suitable for storing infrequently accessed data objects.

Dong *et al.* [12] proposed a cost-effective and practical storage strategy for deciding whether a generated data set should be stored or not at runtime in the cloud. The main focus of this strategy is the local optimization

for the trade-off between computation and storage, while also taking user preferences on storage into account. Khanafer *et al.* [13] introduced a variant of the classical ski-rental problem in which the skier knows the first or second moment of the distribution of ski days. They applied the solution of this variant to the cloud storage system. Scouarnec *et al.* [14] studied policies for caching data generated from computation to trade the storage cost for the compute cost in cloud-based systems, which takes into consideration the frequency of consumption of a data item and the cloud cost model.

None of the above work, however, addresses the trade-off between storage and transfer in distributed cloud storage. There has also been much work on replica placement in networks with consideration of storage and traffic costs [2, 3]. Nevertheless, this line of work normally assumes a fixed storage cost for placing a replica at each node and does not consider charging the replica storage according to time by “pay-as-you-go” billing.

Veeravalli [15] presented a general model for migrating and caching shared data requested by a set of users geographically separated in a network of nodes. Wang *et al.* [16, 17] proposed offline solutions for a homogeneous cost model in which the storage and transfer costs are identical over the entire network. They also presented a centralized online algorithm [17] that requires global knowledge of all the cached copies in the network. Our work is inspired by the above studies. We make substantial technical advancements by solving the cost optimization problem and developing practical distributed online algorithms for a heterogeneous cost model originating from the real cloud storage charging schemes.

Chapter 3

System Model and Definitions

We consider a distributed cloud storage system including a set of sites $\{s_1, s_2, \dots, s_m\}$, where m is the number of sites. A data object is hosted by the distributed storage system. The object can be replicated and stored at one or multiple sites. For each site s_x , the storage cost to hold a replica of the object at s_x is $u[s_x]$ per time unit.¹ Different sites may have different storage costs. The object can be transferred between sites for creating replicas or serving requests. The transfer cost of the object between any two sites is identical and is denoted as w .

Requests are made to the storage system to retrieve the data object. Each request is sent to one of the sites in the storage system (for example, the nearest site to the user issuing the request). If the site receiving a request holds a replica, the request is served locally. Otherwise, the object has to be transferred from a site holding a replica to the receiving site in order to serve the request. The receiving site may, as a result of the transfer, create a replica of the object and hold it for some period of time. We denote the

¹For clarity, we shall use square brackets to represent parameters given in the problem definition and use round brackets to represent values to be computed in the algorithms.

Table 3.1: Summary of notations

Notation	Definition
s_x	a site in the distributed storage system
$u[s_x]$	the storage cost rate of site s_x
r_i	the i -th request to retrieve the data object
$s[r_i]$	the site receiving request r_i
$t[r_i]$	the time when request r_i arrives
w	the transfer cost of the data object between two sites
$p(i)$	the index of the previous request of r_i at the same site
$b(g)$	the minimum cost to serve request r_g
$B(k, i)$	$\sum_{g=k+1}^{i-1} b(g)$

sequence of requests made to the storage system as $\langle r_1, r_2, \dots, r_n \rangle$, where n is the number of requests. For each request r_i , let $s[r_i]$ denote the site receiving the request and $t[r_i]$ denote the time when the request arrives. For simplicity, we shall assume that all requests arrive at distinct times, i.e., $t[r_1] < t[r_2] < \dots < t[r_n]$.² Table 3.1 summarizes the notations (including some that will be introduced in later chapters for easy reference).

We are interested in determining the replicas of the object to create and hold over time and the transfers to carry out in order to serve all the requests. We refer to the sets of replicas and transfers involved as *a replication strategy*. In order to preserve the object, a mandatory requirement for the replication strategy is that at any time, at least one copy (replica) of the object must be stored in the system. The objective of our problem is to find the replication strategy with the minimum total cost of storage and transfer. Without loss of generality, we shall assume that a copy of the object is initially located at the site $s[r_1]$ of the first request r_1 at time $t[r_1]$.³

Our problem can be visualized by a space-time diagram [17] as illustrated

²If there are multiple requests arriving at the same time, we can assume that their arrivals are separated by infinitesimals.

³If the object is initially located at a site s_{init} other than $s[r_1]$, we can add a hypothetical request to s_{init} arriving an infinitesimal earlier than r_1 .

Figure 3.1: An example of space-time diagram

by Figure 3.1. Each site is associated with a horizontal axis representing the timeline. Each request is represented by a point on the axis of the receiving site. A replica held by a site is represented by a horizontal edge from its creation time to its deletion time. A transfer between two sites is represented by a vertical edge from the source site to the destination site.⁴ In the example of Figure 3.1, there are three sites s_1 , s_2 , s_3 and five requests r_1 , r_2 , r_3 , r_4 , r_5 . The object is initially located at site s_1 where request r_1 arrives. s_1 holds the object until request r_2 arrives and transfers it to site s_2 to serve r_2 . Then a replica is created at s_2 , while the replica at s_1 is removed. When request r_3 arrives, another transfer is carried out from site s_2 to s_3 to serve r_3 . After that, s_2 continues to hold its replica. In addition, a new replica is created at s_3 . Finally, requests r_4 and r_5 are served locally by the replicas at s_2 and s_3 respectively. In the space-time diagram, the cost of a replica edge is the edge length (duration) multiplied by the storage cost rate of the site. The cost of a transfer edge is w . Our problem aims to find a set of replica and transfer edges connecting all the request points with the total cost minimized.

⁴For simplicity, the transfer delay is assumed to be negligible. This is a reasonable assumption for many applications where data objects are not huge or the use/processing of data can commence before the complete data is received such as video streaming, query processing, and data analytics.

Chapter 4

An Optimal Offline Algorithm

We start by computing an optimal replication strategy for a given sequence of requests. An offline algorithm is useful when requests have repeated or predictable patterns.

4.1 Format of Optimal Replication Strategy

We first study the format of an optimal replication strategy.

Proposition 1. *There exists an optimal replication strategy in which for each transfer, there is a request at either the source site or the destination site.*

Proof. Since the data object is initially located at the site $s[r_1]$ receiving the first request r_1 , no transfer is needed to serve r_1 . Thus, we consider only transfers done after time $t[r_1]$.

Suppose that a transfer from a site s_x to a site s_y is carried out at time t in an optimal replication strategy, and there is no request at sites s_x and s_y at time t . It can be inferred that either (1) a replica is held by s_x before the

transfer, or (2) s_x just receives the data object from another site s_z at time t . In the latter case, since there is no request at s_x at time t , the transfer from s_x to s_y can be replaced by a transfer from s_z to s_y without affecting the cost of the replication strategy. If s_z does not have a local request at time t and does not hold a replica before time t , we can find another site transferring to s_z at time t and repeat the replacement until a source site of the transfer having a local request at time t or holding a replica before time t is found. So without loss of generality, we assume that s_x holds a replica before the transfer.

Similarly, since there is no request at s_y at time t , if s_y does not hold a replica after the transfer, s_y must be sending the data object to another site s_z at time t (otherwise, the transfer from s_x to s_y can be removed, which contradicts the optimality of the replication strategy). We can then replace the transfers from s_x to s_y and from s_y to s_z by a single transfer from s_x to s_z without affecting the ability of the replication strategy to serve all the requests, which again contradicts the optimality of the replication strategy. Thus, s_y must hold a replica after the transfer.

If s_x continues to hold the replica after the transfer, we can delay the transfer from s_x to s_y by some small $\varepsilon > 0$ and create the replica at s_y at time $t + \varepsilon$ to save the storage cost. This would not affect the feasibility of the replication strategy since all the transfers originating from s_y during the period $(t, t + \varepsilon)$ can originate from s_x instead. As a result, it contradicts the optimality of the replication strategy.

Now suppose that s_x does not hold the replica after the transfer. If the storage cost rates $u[s_x] > u[s_y]$, we can bring the transfer from s_x to s_y earlier to time $t - \varepsilon$, remove the replica at s_x and add a replica at s_y during

the period $(t - \varepsilon, t)$ to save cost. Similarly, if $u[s_x] < u[s_y]$, we can delay the transfer to time $t + \varepsilon$, extend the replica at s_x and remove the replica at s_y during $(t, t + \varepsilon)$ to save cost. In both cases, it contradicts the optimality of the replication strategy.

Finally, consider the case $u[s_x] = u[s_y]$. We look for the last local request of s_x served by the replica at s_x before time t . Let t' be the time of this request. Then we can bring the transfer from s_x to s_y earlier to time t' , remove the replica at s_x and add a replica at s_y during the period (t', t) . This would not affect the cost of the replication strategy. On the other hand, if the replica at s_x does not serve any local request, we let t' be the creation time of the replica. At time t' , there must be a transfer from another site s_z to s_x . We can replace the transfer from s_x to s_y at time t by a transfer from s_z to s_y at time t' , remove the replica at s_x and add a replica at s_y during (t', t) without affecting the cost of the replication strategy. If there is no request at s_z at time t' , the above procedure can be repeated to either derive a contradiction or bring the transfer from s_z to s_y even earlier until a request at the source site is found. $\square$

Remark. [15] made a weaker claim that the set of data copies can only change at the time instants of requests. [17] made a stronger claim that each transfer must end with a request at the destination site under a homogeneous model in which all sites have the same storage cost rate. As shall be seen later, this does not necessarily hold when the sites can have different storage cost rates. Moreover, none of the above work gave a complete proof of the claim. Our Proposition 1 is critical to the analysis of request types below.

In what follows, we shall consider only the replication strategies having the property stated in Proposition 1. Recall that we aim to find a set

of replica and transfer edges in the space-time diagram to connect all the request points. Given an optimal replication strategy, for each request r , we can trace back along the replica edges (from right to left) and transfer edges (from destination to source) to look for an earlier request that provides the data object to serve r . We refer to this earlier request as the *origin* of r . Note that in an optimal replication strategy, there cannot be a replica edge and a transfer edge both going towards the same request point in the space-time diagram, since otherwise we can remove the transfer to save cost. Thus, starting from a request, there can only be one edge for us to trace back. Similarly, in an optimal replication strategy, there cannot be two transfers going to the same site at the same time. Therefore, the replica (horizontal) edges and transfer (vertical) edges must be alternated along the path from r 's origin to r in the space-time diagram. We can divide all the requests into the following four categories based on how the replica and transfer edges are interleaved between the request and its origin.

To facilitate presentation, for each request r_i , let $p(i)$ denote the index of the previous request of r_i arriving at the same site $s[r_i]$. For example, in Figure 3.1, $p(4) = 2$ (since r_2 and r_4 arrive in succession at site s_2). If a request r_i is the first request arriving at site $s[r_i]$, we define $p(i) = 0$. Note that all the $p(i)$ values can be precomputed by scanning the request sequence once in $O(n)$ time.

Type-D Requests. The edge connecting to a request r_i is a replica edge. When tracing back, we can find another request at site $s[r_i]$ along the replica edge (see Figure 4.1a). That is, site $s[r_i]$ holds a replica from the previous request $r_{p(i)}$ to the request r_i . In this case, the origin of r_i is $r_{p(i)}$. As a special case, we also classify the first request r_1 as a type-D request since

Figure 4.1: Four types of requests

the object is available locally.

Type-E Requests. The edge connecting to a request r_i is a replica edge, but there is no other request at site $s[r_i]$ along the replica edge. Let τ be the creation time of the replica at site $s[r_i]$. Then $s[r_i]$ must be receiving a transfer from another site s_x at time τ (see Figure 4.1b). Since $s[r_i]$ does not have a local request at time τ , by Proposition 1, site s_x must have a local request r_j at time τ . In this case, the origin of r_i is r_j .

Type-F Requests. The edge connecting to a request r_i is a transfer edge, i.e., site $s[r_i]$ receives a transfer from another site s_x at the time $t[r_i]$ of r_i . The transfer edge must be connected to a replica edge at site s_x . When tracing back, we can find a request r_j at s_x along the replica edge (see Figure 4.1c). That is, s_x holds a replica from time $t[r_j]$ to $t[r_i]$ and then transfers the object to site $s[r_i]$ to serve r_i . In this case, the origin of r_i is r_j . We remark that both type-E and type-F requests are possible in an optimal replication strategy. This is because different sites can have different storage costs. If s_x has a lower storage cost rate than $s[r_i]$, it is beneficial to keep a replica

at s_x for saving cost. On the other hand, if s_x has a higher storage cost rate than $s[r_i]$, it is favourable to keep a replica at $s[r_i]$ for saving cost.

Type-G Requests. The edge connecting to a request r_i is a transfer edge from site s_x . The transfer edge is connected to a replica edge at s_x , but there is no request at s_x along the replica edge. Let τ be the creation time of the replica at s_x . Then s_x must be receiving a transfer from another site s_y at time τ (see Figure 4.1d). Since s_x does not have a local request at time τ , by Proposition 1, s_y must have a local request r_j at time τ . In this case, the origin of r_i is r_j . We remark that type-G requests are possible in an optimal replication strategy also because different sites can have different storage costs. If the period $(\tau, t[r_i])$ is quite long and site s_x has a storage cost rate much lower than both sites $s[r_i]$ and s_y , it is helpful to transfer the object to s_x and keep a replica there for saving cost, even if the replica does not serve any local request at s_x and the object needs to be transferred again later to $s[r_i]$ for serving r_i . Note that $s[r_i]$ and s_y could even be the same site for type-G requests.

4.2 A Dynamic Programming Algorithm

We develop a dynamic programming algorithm to compute an optimal replication strategy. Consider a sequence of requests $\langle r_1, r_2, \dots, r_n \rangle$. Define $C(i)$ as the cost of an optimal replication strategy to serve the subsequence of the first i requests, i.e., $\langle r_1, r_2, \dots, r_i \rangle$. We also define $D(i)/E(i)/F(i)/G(i)$ as the cost of the best replication strategy to serve $\langle r_1, r_2, \dots, r_i \rangle$ subject to the constraint that the last request r_i is a type-D/E/F/G request. By the

analysis above, we have for each $1 \leq i \leq n$,

$$C(i) = \min\{D(i), E(i), F(i), G(i)\}. \quad (4.1)$$

4.2.1 Preliminaries

We first derive some useful features of an optimal replication strategy, which are inspired by [17]. The following feature says that if two successive requests at the same site are sufficiently close in time, the site should hold a replica between them.

Proposition 2. *For each request r_i ($i > 1$), if $u[s[r_i]] \cdot (t[r_i] - t[r_{p(i)}]) < w$, in any optimal replication strategy, site $s[r_i]$ must hold a replica during the period $(t[r_{p(i)}], t[r_i])$.*

Proof. If site $s[r_i]$ does not hold a replica throughout the period $(t[r_{p(i)}], t[r_i])$, to serve r_i , site $s[r_i]$ must either (1) receive a transfer from another site at time $t[r_i]$, or (2) receive a transfer at some time instant between requests $r_{p(i)}$ and r_i and then hold a replica till time $t[r_i]$. In either case, the transfer cost incurred is w . If the transfer is replaced by a replica from $r_{p(i)}$ to r_i at $s[r_i]$, the total cost can be reduced, which contradicts the optimality of the replication strategy. $\square$

By the reasoning above, serving each request r_i ($i > 1$) requires either maintaining a replica at $s[r_i]$ throughout the period $(t[r_{p(i)}], t[r_i])$ or a transfer to $s[r_i]$ between $r_{p(i)}$ and r_i . On the other hand, if $p(i) = 0$ (r_i is the first request arriving at a site), a transfer is indispensable to serve r_i . Define

$$b(g) = \begin{cases} \min\{u[s[r_g]] \cdot (t[r_g] - t[r_{p(g)}]), w\} & \text{if } p(g) \neq 0, \\ w & \text{if } p(g) = 0. \end{cases}$$

Figure 4.2: A replica at site s_x crossing time $t[r_j]$

We have the following bound on the optimal cost.

Proposition 3. *The cost of an optimal replication strategy for $\langle r_1, r_2, \dots, r_n \rangle$ is at least $\sum_{g=2}^n b(g)$.*

Note, however, that the bound $\sum_{g=i}^j b(g)$ does not always apply to a subsequence of requests $\langle r_i, r_{i+1}, \dots, r_j \rangle$. For example, in Figure 3.1, the additional cost to serve request r_4 on top of the replication strategy for $\langle r_1, r_2, r_3 \rangle$ is $u[s_2] \cdot (t[r_4] - t[r_3])$, which can be lower than $b(4)$ if requests r_3 and r_4 are close in time. We need extra conditions to establish the bound for a subsequence of requests.

Suppose that in an optimal replication strategy for $\langle r_1, r_2, \dots, r_i \rangle$, site $s[r_i]$ holds a replica during the period $(t[r_j], t[r_i])$, where r_j is a request with index j satisfying $p(i) \leq j < i$ (see Figure 4.2a for an illustration).

We say that a replica *crosses* a time instant t if the replica is created strictly before t and deleted strictly after t . It is easy to show that for any site $s_x \neq s[r_i]$, if s_x holds a replica crossing time $t[r_j]$, the replica must serve at least one local request after $t[r_j]$. In fact, if it does not serve any local request (see Figure 4.2a for an illustration), the replica can be deleted earlier without affecting the ability of the replication strategy to serve all requests. This is because all the transfers originating from the replica at s_x after $t[r_j]$ can originate from the replica at $s[r_i]$ instead (see Figure 4.2b

Figure 4.3: r_g and $r_{p(g)}$

for an illustration). As a consequence, it contradicts the optimality of the replication strategy. Thus, each replica crossing time $t[r_j]$ must be kept till at least the first local request after $t[r_j]$.

Among the requests between r_j and r_i , define $X(j)$ as the set of all the first requests after time $t[r_j]$ at sites other than $s[r_j]$, i.e., $X(j) = \{r_l \mid p(l) < j < l < i\}$. Note that by definition, all the requests in $X(j)$ are from different sites. We further define $X_S(j)$ as the subset of $X(j)$ including those requests satisfying $u[s[r_l]] \cdot (t[r_l] - t[r_{p(l)}]) < w$. For each request $r_l \in X_S(j)$, by Proposition 2, there must be a replica at site $s[r_l]$ during the period $(t[r_{p(l)}], t[r_l])$ and this replica crosses time $t[r_j]$. Among all the requests in $X(j)$, we find the highest-indexed request r_k such that there is a replica at site $s[r_k]$ crossing time $t[r_j]$. Then the set of possible requests for r_k is given by $X^*(j) = \{r_k \mid r_k \in X(j) \text{ and } k \geq \max_{r_l \in X_S(j)} l\}$. In the case that $X_S(j) = \emptyset$, we define $X^*(j) = X(j)$.

Next, we show that given r_k , the minimum achievable cost to serve the requests $\langle r_{k+1}, r_{k+2}, \dots, r_{i-1} \rangle$ is $\sum_{g=k+1}^{i-1} b(g)$.

Consider each request r_g where $k+1 \leq g \leq i-1$. If $p(g) = 0$ (r_g is the first request arriving at a site), a transfer is essential to serve r_g and

thus the minimum achievable cost is w . Now consider the case $p(g) > j$ (see Figure 4.3a for an illustration). If $u[s[r_g]] \cdot (t[r_g] - t[r_{p(g)}]) < w$, by Proposition 2, there must be a replica at site $s[r_g]$ during the period $(t[r_{p(g)}], t[r_g])$ and hence the cost to serve r_g is $u[s[r_g]] \cdot (t[r_g] - t[r_{p(g)}])$. If $u[s[r_g]] \cdot (t[r_g] - t[r_{p(g)}]) \geq w$, to serve r_g , site $s[r_g]$ must either hold a replica throughout the period $(t[r_{p(g)}], t[r_g])$ or receive a transfer at some time instant between $r_{p(g)}$ and r_g . Since there is a replica at $s[r_i]$, the best strategy is to conduct a transfer from $s[r_i]$ to $s[r_g]$ at time $t[r_g]$, which gives the minimum achievable cost of w . In summary, the minimum achievable cost to serve r_g is $\min\{u[s[r_g]] \cdot (t[r_g] - t[r_{p(g)}]), w\}$. Finally, consider the case $0 < p(g) < j < g$, i.e., $g \in X(j)$ (see Figure 4.3b for an illustration). Since $g > k \geq \max_{r_l \in X_S(j)} l$, by definition, we must have $u[s[r_g]] \cdot (t[r_g] - t[r_{p(g)}]) \geq w$. Moreover, by the definition of k , there is no replica at site $s[r_g]$ crossing time $t[r_g]$. This implies that a transfer to site $s[r_g]$ at some time instant between r_j and r_g is indispensable in order to serve s_g . Thus, the best strategy is again to conduct a transfer from $s[r_i]$ to $s[r_g]$ at time $t[r_g]$, which gives the minimum achievable cost of $w = \min\{u[s[r_g]] \cdot (t[r_g] - t[r_{p(g)}]), w\}$.

By similar arguments, if r_k does not exist (i.e., there is no replica crossing time $t[r_j]$ at sites other than $s[r_j]$), which can only happen if $X_S(j) = \emptyset$, the minimum achievable cost to serve the requests $\langle r_{j+1}, r_{j+2}, \dots, r_{i-1} \rangle$ is $\sum_{g=j+1}^{i-1} b(g)$.

To simplify notations, we define $B(k, i) = \sum_{g=k+1}^{i-1} b(g)$.

Proposition 4. *In an optimal replication strategy for $\langle r_1, r_2, \dots, r_i \rangle$, site $s[r_i]$ holds a replica during the period $(t[r_j], t[r_i])$ for some j satisfying $p(i) \leq j < i$.*

(1) *If r_k is the highest-indexed request in $X^*(j)$ such that there is a*

Figure 4.4: Recurrence for $D(i)$

replica at site $s[r_k]$ crossing time $t[r_j]$, then r_k is served by this replica, and the minimum achievable cost to serve the requests $\langle r_{k+1}, r_{k+2}, \dots, r_{i-1} \rangle$ is $B(k, i)$.

(2) If there is no replica crossing time $t[r_j]$ at sites other than $s[r_j]$, the minimum achievable cost to serve the requests $\langle r_{j+1}, r_{j+2}, \dots, r_{i-1} \rangle$ is $B(j, i)$.

The sets $X(j)$ and hence $X_S(j)$ and $X^*(j)$ for all j 's can be precomputed by scanning the request sequence backwards in $O(mn)$ time, where m is the number of sites and n is the number of requests. The $B(1, i)$ values for all i 's can be precomputed in $O(n)$ time. Then with these values, any $B(k, i)$ value can be computed in $O(1)$ time by $B(1, i) - B(1, k + 1)$.

We remark that if the replica during the period $(t[r_j], t[r_i])$ is held by another site instead of $s[r_j]$, as long as that site does not have any local request during this period, the minimum achievable costs derived in Proposition 4 still hold.

4.2.2 Recurrence for $D(i)$

By definition, if r_i is a type-D request (see Figure 4.1a), site $s[r_i]$ holds a replica from request $r_{p(i)}$ to r_i .

If $X_S(p(i)) = \emptyset$, in an optimal replication strategy for $D(i)$, it is possible that no replica crosses time $t[r_{p(i)}]$ at any site other than $s[r_i]$ (see Figure 4.4a

for an illustration). In this case, the replicas and transfers up to time $t[r_{p(i)}]$ can serve all the requests from r_1 to $r_{p(i)}$. Thus, their total cost is at least $C(p(i))$. On the other hand, by Proposition 4, the minimum achievable cost to serve the requests $\langle r_{p(i)+1}, \dots, r_i \rangle$ is $B(p(i), i)$. In addition, the storage cost of the replica held by site $s[r_i]$ from $r_{p(i)}$ to r_i is $u[s[r_i]] \cdot (t[r_i] - t[r_{p(i)}])$. Therefore, the minimum achievable cost to serve $\langle r_1, r_2, \dots, r_i \rangle$ is

$$C(p(i)) + B(p(i), i) + u[s[r_i]] \cdot (t[r_i] - t[r_{p(i)}]). \quad (4.2)$$

If there is at least one replica crossing time $t[r_{p(i)}]$ at sites other than $s[r_i]$, we find the highest-indexed request $r_k \in X^*(p(i))$ such that there is a replica at site $s[r_k]$ crossing time $t[r_{p(i)}]$ (see Figure 4.4b for an illustration). By the analysis in the preliminaries, r_k is served locally by this replica. Therefore, r_k must be a type-D or type-E request. If r_k is a type-D request, the replica at $s[r_k]$ is created no later than $r_{p(k)}$ and hence earlier than $r_{p(i)}$. Thus, the replica definitely crosses time $t[r_{p(i)}]$. In this case, the total cost for serving the requests from r_1 up to r_k is at least $D(k)$. If r_k is a type-E request, the replica at $s[r_k]$ must be created earlier than $r_{p(i)}$ in order to cross time $t[r_{p(i)}]$. Define $E'(k)$ as the cost of the best strategy to serve $\langle r_1, r_2, \dots, r_k \rangle$ subject to the constraints that r_k is a type-E request and the replica ending at r_k starts earlier than $r_{p(i)}$. Then, the total cost for serving the requests from r_1 up to r_k is at least $E'(k)$. On the other hand, by Proposition 4, the minimum achievable cost to serve the requests $\langle r_{k+1}, \dots, r_i \rangle$ is $B(k, i)$. Therefore, together with the cost of the replica held by site $s[r_i]$ from $r_{p(i)}$ to r_i , the minimum achievable cost to serve $\langle r_1, r_2, \dots, r_i \rangle$ is

$$\min\{D(k), E'(k)\} + B(k, i) + u[s[r_i]] \cdot (t[r_i] - t[r_{p(i)}]). \quad (4.3)$$

Figure 4.5: $E(k)$ and $E'(k)$

Without knowing r_k , we can iterate through all the possible $r_k \in X^*(p(i))$ and choose the lowest value of (4.2) and (4.3) to derive $D(i)$. Then, the recurrence for $D(i)$ is given by

$$D(i) = u[s[r_i]] \cdot (t[r_i] - t[r_{p(i)}]) + \min \left\{ \begin{array}{l} C(p(i)) + B(p(i), i) \text{ (included only if } X_S(p(i)) = \emptyset \text{)} \\ \min_{r_k \in X^*(p(i))} \{D(k) + B(k, i)\} \\ \min_{r_k \in X^*(p(i))} \{E'(k) + B(k, i)\} \end{array} \right\}. \quad (4.4)$$

Note that by definition, we have $E'(k) \geq E(k)$. Next, we prove that for each $r_k \in X^*(p(i))$, if $E'(k) > E(k)$, then

$$E(k) + B(k, i) + u[s[r_i]] \cdot (t[r_i] - t[r_{p(i)}]) > D(i). \quad (4.5)$$

In fact, if $E'(k) > E(k)$, the optimal replication strategies for $E(k)$ and $E'(k)$ are different. It follows from the definition of $E'(k)$ that: in the optimal replication strategy for $E(k)$, the replica ending at r_k starts no earlier than $r_{p(i)}$ (see Figure 4.5a for an illustration). That is, denoting r_h as the origin of r_k , we have $t[r_h] \geq t[r_{p(i)}]$. Then we can remove the replica ending at r_k as well as the transfer that creates the replica, and add a transfer from site $s[r_i]$ to $s[r_k]$ to serve r_k (see Figure 4.5b for an illustration). This reduces the total cost to serve the requests and hence (4.5) holds. Thus, the computation of

$D(i)$ would not be affected if we replace $E'(k)$ with $E(k)$ for those indexes k satisfying $E'(k) > E(k)$. On the other hand, we can certainly replace $E'(k)$ with $E(k)$ for those indexes k satisfying $E'(k) = E(k)$. As a result, we can rewrite the recurrence (4.4) as

$$D(i) = u[s[r_i]] \cdot (t[r_i] - t[r_{p(i)}]) + \min \left\{ \begin{array}{l} C(p(i)) + B(p(i), i) \text{ (included only if } X_S(p(i)) = \emptyset) \\ \min_{r_k \in X^*(p(i))} \{D(k) + B(k, i)\} \\ \min_{r_k \in X^*(p(i))} \{E(k) + B(k, i)\} \end{array} \right\}.$$

Note that $X^*(p(i))$ includes at most one request at each site. Thus, the time complexity to compute $D(i)$ for a given i is $O(m)$. So, the total time complexity to compute $D(i)$ for all i 's is $O(mn)$. Recall that r_i can be a type-D request only if there is a previous request $r_{p(i)}$ at the same site (except for r_1). For each request r_i ($i > 1$) that is the first request arriving at site $s[r_i]$, we define $D(i) = +\infty$. For request r_1 , $D(1) = 0$.

4.2.3 Recurrence for $E(i)$

If r_i is a type-E request (see Figure 4.1b), the origin of r_i is a request r_j at another site $s[r_j]$ and $p(i) < j < i$.

First, we prove that in an optimal replication strategy for $E(i)$, there is no replica crossing time $t[r_j]$ at any site other than $s[r_j]$. Assume on the contrary that there is a replica crossing time $t[r_j]$ at a site $s_x \neq s[r_j]$. Suppose that this replica ends at time τ (see Figure 4.6a for an illustration). Then, we can replace the transfer from $s[r_j]$ to $s[r_i]$ at time $t[r_j]$ by a transfer from s_x to $s[r_i]$ at time τ , and delay the creation of the replica at $s[r_i]$ to time τ . All the transfers originating from $s[r_i]$ during the period $(t[r_j], \tau)$

Figure 4.6: A replica crosses time $t[r_j]$

can originate from s_x instead with the same cost (see Figure 4.6b for an illustration). As such, we can reduce the total cost to serve all the requests $\langle r_1, r_2, \dots, r_i \rangle$ while maintaining r_i as a type-E request, which contradicts the optimality of the replication strategy.

Since there is no replica crossing time $t[r_j]$ at any site other than $s[r_j]$, we must have $X_S(j) = \emptyset$. Moreover, the replicas and transfers up to time $t[r_j]$ can serve all the requests from r_1 to r_j . Hence, their total cost is at least $C(j)$. By Proposition 4, the minimum achievable cost to serve the requests $\langle r_{j+1}, \dots, r_{i-1} \rangle$ is $B(j, i)$. In addition, the cost of the path from r_j to r_i in the space-time diagram is $u[s[r_i]] \cdot (t[r_i] - t[r_j]) + w$. Therefore, the minimum achievable cost to serve $\langle r_1, r_2, \dots, r_i \rangle$ is

$$C(j) + B(j, i) + u[s[r_i]] \cdot (t[r_i] - t[r_j]) + w. \quad (4.6)$$

Without knowing r_j , we can iterate through all possible j values satisfying $p(i) < j < i$ and $X_S(j) = \emptyset$, and choose the lowest value of (4.6) to derive $E(i)$. Then, the recurrence for $E(i)$ is given by

$$E(i) = \min_{\substack{p(i) < j < i \\ X_S(j) = \emptyset}} \{C(j) + B(j, i) + u[s[r_i]] \cdot (t[r_i] - t[r_j]) + w\}.$$

Figure 4.7: Recurrence for $F(i)$

For a given i , the number of valid j values is bounded by $i - p(i)$. Thus, for all the requests r_i at the same site, the total number of valid j values is bounded by n (the total number of requests). So, the total time complexity to compute $E(i)$ for all i 's is $O(mn)$, where m is the number of sites. For request r_1 , we define $E(1) = +\infty$, since r_1 is a type-D request.

4.2.4 Recurrence for $F(i)$

By definition, if r_i is a type-F request (see Figure 4.1c), the origin of r_i is a request r_j at another site $s[r_j]$, and r_j is the last request at $s[r_j]$ before time $t[r_i]$.

If $X_S(j) = \emptyset$, in an optimal replication strategy for $F(i)$, it is possible that no replica crosses time $t[r_j]$ at any site other than $s[r_j]$ (see Figure 4.7a for an illustration). In this case, the replicas and transfers up to time $t[r_j]$ can serve all the requests from r_1 to r_j . Thus, their total cost is at least $C(j)$. On the other hand, by Proposition 4, the minimum achievable cost to serve the requests $\langle r_{j+1}, \dots, r_{i-1} \rangle$ is $B(j, i)$. In addition, the cost of the path from r_j to r_i in the space-time diagram is $u[s[r_j]] \cdot (t[r_i] - t[r_j]) + w$. Therefore, the minimum achievable cost to serve $\langle r_1, r_2, \dots, r_i \rangle$ is

$$C(j) + B(j, i) + u[s[r_j]] \cdot (t[r_i] - t[r_j]) + w. \quad (4.7)$$

If there is at least one replica crossing time $t[r_j]$ at sites other than $s[r_j]$, we find the highest-indexed request $r_k \in X^*(j)$ such that there is a replica at site $s[r_k]$ crossing time $t[r_j]$ (see Figure 4.7b for an illustration). By the analysis in the preliminaries, r_k is served locally by this replica. Therefore, r_k must be a type-D or type-E request. By similar arguments to the case of $D(i)$, the minimum achievable cost to serve $\langle r_1, r_2, \dots, r_i \rangle$ is

$$\min\{D(k), E'(k)\} + B(k, i) + u[s[r_j]] \cdot (t[r_i] - t[r_j]) + w. \quad (4.8)$$

Without knowing r_j and r_k , we can iterate through all their possible values and choose the lowest value of (4.7) and (4.8) to derive $F(i)$. Recall that r_j is the last request at $s[r_j]$ before time $t[r_i]$, and $r_k \in X^*(j)$. For each site $s_x \neq s[r_i]$, we define $q_x(i)$ as the index of the last request at s_x before time $t[r_i]$ (if there is no request at s_x before time $t[r_i]$, we define $q_x(i) = \perp$ and such site s_x does not need to be considered). Then, the recurrence for $E(i)$ is given by

$$F(i) = \min_{\substack{s_x \neq s[r_i] \\ q_x(i) \neq \perp}} \left\{ u[s_x] \cdot (t[r_i] - t[r_{q_x(i)}]) + w \right. \\ \left. + \min \left\{ \begin{array}{l} C(q_x(i)) + B(q_x(i), i) \text{ (included only if } X_S(j) = \emptyset) \\ \min_{r_k \in X^*(q_x(i))} \{D(k) + B(k, i)\} \\ \min_{r_k \in X^*(q_x(i))} \{E'(k) + B(k, i)\} \end{array} \right\} \right\}.$$

Again, by similar arguments to the case of $D(i)$, we can replace $E'(k)$ with $E(k)$ in the above recurrence.

Given a site $s_x \neq s[r_i]$, there is exactly one $q_x(i)$, and $X^*(q_x(i))$ includes at most one request at each site. Thus, the time complexity to compute

$F(i)$ for a given i is $O(m^2)$. So, the total time complexity to compute $F(i)$ for all i 's is $O(m^2n)$. For request r_1 , we define $F(1) = +\infty$, since r_1 is a type-D request.

4.2.5 Recurrence for $G(i)$

If r_i is a type-G request (see Figure 4.1d), the origin of r_i is a request r_j at the same site or a different site. Site $s[r_j]$ sends the object to a site s_x (other than $s[r_i]$ and $s[r_j]$) at time $t[r_j]$. Then, s_x creates a replica, keeps it till time $t[r_i]$, and sends the object to site $s[r_i]$ to serve r_i . s_x does not have any local request during the period $(t[r_j], t[r_i])$.

Let $s_{\min}$ denote the site that has the lowest storage cost rate (if there are multiple sites sharing the same lowest storage cost, we randomly designate one of these sites as $s_{\min}$). We first show that in an optimal replication strategy for $G(i)$, if $s_x \neq s_{\min}$, such a replication strategy would not affect the computation of $C(i)$. In fact, if $s_x \neq s_{\min}$ (see Figure 4.8a for an illustration), we can replace the transfer from $s[r_j]$ to s_x at time $t[r_j]$ by a transfer from $s[r_j]$ to $s_{\min}$, and replace the transfer from s_x to $s[r_i]$ at time $t[r_i]$ by a transfer from $s_{\min}$ to $s[r_i]$ (see Figure 4.8b for an illustration). All the transfers originating from s_x during the period $(t[r_j], t[r_i])$ can originate from $s_{\min}$ instead with the same cost. In this way, we can reduce the total cost to serve all the requests $\langle r_1, r_2, \dots, r_i \rangle$ (or maintain the same total cost if s_x has the same storage cost rate as $s_{\min}$). Thus, an optimal replication strategy for $G(i)$ in which $s_x \neq s_{\min}$ would not contribute to the computation of $C(i)$. Moreover, $G(i)$ is not used in the recurrences for computing $D(i)/E(i)/F(i)$ earlier. Since our ultimate goal is to compute $C(i)$ according to (4.1), we only need to consider the case of $s_x = s_{\min}$ in computing $G(i)$.

Figure 4.8: $s_x \neq s_{\min}$ for $G(i)$

By similar arguments to the case of $E(i)$, in an optimal replication strategy for $G(i)$, there is no replica crossing time $t[r_j]$ at any site other than $s_{\min}$. Thus, we must have $X_S(j) = \emptyset$. Moreover, the replicas and transfers up to time $t[r_j]$ can serve all the requests from r_1 to r_j . Hence, their total cost is at least $C(j)$. By Proposition 4, the minimum achievable cost to serve the requests $\langle r_{j+1}, \dots, r_{i-1} \rangle$ is $B(j, i)$. In addition, the cost of the path from r_j to r_i in the space-time diagram is $u[s_{\min}] \cdot (t[r_i] - t[r_j]) + 2w$. Therefore, the minimum achievable cost to serve $\langle r_1, r_2, \dots, r_i \rangle$ is

$$C(j) + B(j, i) + u[s_{\min}] \cdot (t[r_i] - t[r_j]) + 2w. \quad (4.9)$$

Without knowing r_j , we can iterate through all possible j values and choose the lowest value of (4.9) to derive $G(i)$. Recall that there is no request at site $s_{\min}$ between time $t[r_j]$ and $t[r_i]$. Let $h(i)$ denote the index of the last request at $s_{\min}$ before time $t[r_i]$ (define $h(i) = 0$ if there is no request at $s_{\min}$ before time $t[r_i]$). Then we have $h(i) < j < i$. So, the recurrence for $G(i)$ is given by

$$G(i) = \min_{\substack{h(i) < j < i \\ X_S(j) = \emptyset}} \{C(j) + B(j, i) + u[s_{\min}] \cdot (t[r_i] - t[r_j]) + 2w\}.$$

In the worst case, the time complexity to compute $G(i)$ for a given i is $O(i)$. Thus, the total time complexity to compute $G(i)$ for all i 's is $O(n^2)$. For request r_1 , we define $G(1) = +\infty$, since r_1 is a type-D request.

After computing $D(i)/E(i)/F(i)/G(i)$ for each i , $C(i)$ can be derived by (4.1). Therefore, the overall time complexity of the dynamic programming algorithm is $O(\max\{m^2n, n^2\})$.

We need to store $X(j)$ to compute $C(i)/D(i)/E(i)/F(i)/G(i)$ for each request r_i . The space complexity of $X(j)$ for all j 's is $O(mn)$. We also need the values of $B(j, i)$. Each $B(j, i)$ can be computed by $B(1, i) - B(1, j + 1)$. The space complexity of $B(1, i)$ for all i 's is $O(n)$. Thus, the overall space complexity is $O(mn)$.

Chapter 5

Online Algorithms

Now, we consider an online setting in which future request arrivals are not known and present two distributed algorithms.

5.1 A Basic Online Algorithm

Since at least one replica must be stored at any time, the cost of any replication strategy is at least the storage cost of a replica at site $s_{\min}$ (the site with the lowest storage cost rate) over the time span.

Proposition 5. *The cost of an optimal replication strategy for $\langle r_1, r_2, \dots, r_n \rangle$ is at least $u[s_{\min}] \cdot (t[r_n] - t[r_1])$.*

The main idea of our basic algorithm is to always keep a replica at site $s_{\min}$ and create replicas at other sites only when necessary. We refer to the replica at $s_{\min}$ as *the master replica*. All the requests arriving at $s_{\min}$ are served locally by the master replica.

Planning the replicas and transfers for other sites resembles a continuous version of the classical ski-rental problem [18] in that there is a trade-off

between holding and not holding a replica at each site. When a request arrives at a site s_x other than $s_{\min}$, a transfer is carried out from site $s_{\min}$ to s_x to serve the request. On receiving the transfer, site s_x may create a replica and hold it for some time period. The storage cost for holding the replica is $u[s_x]$ per time unit. If a replica is not available at site s_x when the next request arrives at s_x , a transfer needs to be conducted from site $s_{\min}$ to s_x again to serve the request, which incurs a cost of w . To minimize the cost incurred for serving the next request, site s_x should hold a replica if the next request will arrive within a period of $\frac{w}{u[s_x]}$ and should not hold a replica otherwise. In the online setting, site s_x does not know when the next request will arrive. Thus, it can hold a replica for a period of $\frac{w}{u[s_x]}$ on receiving a transfer so that the accumulated storage cost is equal to the transfer cost. If the next request arrives at s_x before this period ends, s_x renews the replica and holds it for another period of $\frac{w}{u[s_x]}$. Otherwise, if no request arrives at s_x during the period of $\frac{w}{u[s_x]}$, s_x deletes the replica and a transfer will be carried out to serve the next request at s_x . Algorithm 1 shows the detailed algorithm executed by a site s_x other than $s_{\min}$.

Algorithm 1 Basic online algorithm executed by a site $s_x \neq s_{\min}$

- 1: *Initialization (at time t_0):*
 - 2: $t_{\text{expire}} \leftarrow t_0$; //expiration time of local replica
 - 3: *On receiving a request at time t :*
 - 4: **if** $t > t_{\text{expire}}$ **then**
 - 5: transfer object from site $s_{\min}$;
 - 6: create local replica;
 - 7: **end if**
 - 8: $t_{\text{expire}} \leftarrow t + w/u[s_x]$;
 - 9: serve the request with local replica;
 - 10: *On expiration of local replica at time t_{expire} :*
 - 11: delete local replica;
-

Theorem 1. *The basic online algorithm is 3-competitive.*

Proof. Let OPT denote the cost of an optimal replication strategy for a sequence of requests $\langle r_1, r_2, \dots, r_n \rangle$.

In the basic online algorithm, the storage cost at site $s_{\min}$ is $u[s_{\min}] \cdot (t[r_n] - t[r_1]) \leq \text{OPT}$, where the inequality follows from Proposition 5.

For each site $s_x \neq s_{\min}$, let $\langle r_1^x, r_2^x, r_3^x, \dots \rangle$ be the subsequence of requests arriving at s_x . Among these requests, let $\langle r_{i_1}^x, r_{i_2}^x, r_{i_3}^x, \dots \rangle$ be the subsequence of requests that results in transfers from $s_{\min}$ to s_x . For each $j \geq 1$, s_x holds a replica from time $t[r_{i_j}^x]$ to $t[r_{i_{j+1}-1}^x] + \frac{w}{u[s_x]}$. Thus, the cost incurred after request $r_{i_j}^x$ till request $r_{i_{j+1}}^x$ is given by

$$\begin{aligned} & w + u[s_x] \cdot (t[r_{i_{j+1}-1}^x] + \frac{w}{u[s_x]} - t[r_{i_j}^x]) \\ &= 2w + u[s_x] \cdot (t[r_{i_{j+1}-1}^x] - t[r_{i_j}^x]). \end{aligned}$$

By the algorithm definition, the inter-arrival time between any two successive requests r_k^x and r_{k+1}^x (where $i_j \leq k < k+1 \leq i_{j+1}-1$) is at most $\frac{w}{u[s_x]}$, and the inter-arrival time between $r_{i_{j+1}-1}^x$ and $r_{i_{j+1}}^x$ is longer than $\frac{w}{u[s_x]}$. Thus,

$$\begin{aligned} & 2w + u[s_x] \cdot (t[r_{i_{j+1}-1}^x] - t[r_{i_j}^x]) \\ &= 2 \cdot \min\{u[s_x] \cdot (t[r_{i_{j+1}}^x] - t[r_{i_{j+1}-1}^x]), w\} + \sum_{k=i_j}^{i_{j+1}-2} \min\{u[s_x] \cdot (t[r_{k+1}^x] - t[r_k^x]), w\} \\ &\leq 2 \cdot \sum_{k=i_j}^{i_{j+1}-1} \min\{u[s_x] \cdot (t[r_{k+1}^x] - t[r_k^x]), w\}. \end{aligned}$$

Note that the first request r_1^x arriving at s_x is served by a transfer with cost

w . Hence, the total cost involving s_x is bounded by

$$w + 2 \cdot \sum_{k \geq 1} \min\{u[s_x] \cdot (t[r_{k+1}^x] - t[r_k^x]), w\},$$

and the total cost involving all sites $s_x \neq s_{\min}$ is bounded by $2 \cdot \sum_{g=2}^n b(g) \leq 2 \cdot \text{OPT}$, where the inequality follows from Proposition 3.

Therefore, the total cost of the basic online algorithm is bounded by $3 \cdot \text{OPT}$. $\square$

5.2 A Dynamic Online Algorithm

In the basic online algorithm, site $s_{\min}$ holds the master replica all the time to fulfill the requirement that at least one replica must be available at any time. If there are much less requests arriving at $s_{\min}$ than some other site s_x , it may be beneficial to keep the master replica at s_x rather than $s_{\min}$ for saving cost. This is because the replica can eliminate more transfers to s_x than those to $s_{\min}$, which may outweigh the increase in the storage cost.

Assume that request arrivals at a site s_x follow a Poisson process. Let $\lambda[s_x]$ denote the request rate to s_x . Then the inter-arrival time θ between two successive requests to s_x follows an exponential distribution with a probability density function of $\lambda[s_x] \cdot e^{-\lambda[s_x] \cdot \theta}$ and a mean of $\frac{1}{\lambda[s_x]}$. Suppose that s_x does not hold the master replica. If $\theta > \frac{w}{u[s_x]}$, to serve the incoming request, we need to transfer the data object from the master replica to s_x . After that, a local replica is created at s_x and kept for a time period of $\frac{w}{u[s_x]}$. Thus, the total cost is $w + u[s_x] \cdot \frac{w}{u[s_x]} = 2w$. On the other hand, if $\theta \leq \frac{w}{u[s_x]}$, the incoming request can be served by the local replica directly, so the cost is $u[s_x] \cdot \theta$. Therefore, the expected cost for serving a request arriving at site

(a) Cost saving with respect to u and w when $\lambda = 1$

(b) Cost saving with respect to u and λ when $w = 1$

(c) Cost saving with respect to w and u when $\lambda = 1$

(d) Cost saving with respect to w and λ when $u = 1$

(e) Cost saving with respect to λ and w when $u = 1$

(f) Cost saving with respect to λ and u when $w = 1$

Figure 5.1: Numerical results of cost saving

s_x is

$$\begin{aligned}
& \int_{\frac{w}{u[s_x]}}^{+\infty} 2w \cdot (\lambda[s_x] \cdot e^{-\lambda[s_x] \cdot \theta}) d\theta + \int_0^{\frac{w}{u[s_x]}} (u[s_x] \cdot \theta) \cdot (\lambda[s_x] \cdot e^{-\lambda[s_x] \cdot \theta}) d\theta \\
&= 2w \cdot e^{\frac{-\lambda[s_x] \cdot w}{u[s_x]}} + u[s_x] \cdot \lambda[s_x] \cdot \left(\left(-\frac{w}{u[s_x] \cdot \lambda[s_x]} - \frac{1}{(\lambda[s_x])^2} \right) \cdot e^{\frac{-\lambda[s_x] \cdot w}{u[s_x]}} + \frac{1}{(\lambda[s_x])^2} \right) \\
&= \frac{u[s_x]}{\lambda[s_x]} + \left(w - \frac{u[s_x]}{\lambda[s_x]} \right) \cdot e^{\frac{-\lambda[s_x] \cdot w}{u[s_x]}}.
\end{aligned}$$

Taking into account the request rate $\lambda[s_x]$, the cost incurred for site s_x per time unit is

$$u[s_x] + (\lambda[s_x] \cdot w - u[s_x]) \cdot e^{\frac{-\lambda[s_x] \cdot w}{u[s_x]}}.$$

If s_x holds the master replica, no transfer is needed to serve the requests arriving at s_x , so the cost incurred is $u[s_x]$ per time unit. As a result, the cost saving per time unit for keeping the master replica at s_x is given by

$$\begin{aligned}
& \left(u[s_x] + (\lambda[s_x] \cdot w - u[s_x]) \cdot e^{\frac{-\lambda[s_x] \cdot w}{u[s_x]}} \right) - u[s_x] \\
&= (\lambda[s_x] \cdot w - u[s_x]) \cdot e^{\frac{-\lambda[s_x] \cdot w}{u[s_x]}}.
\end{aligned}$$

Figure 5.1 shows some numerical results of the cost saving for different $u[s_x]$, $\lambda[s_x]$ and w values. In each figure, we fix one parameter, change another parameter along the x-axis, and plot multiple curves for different values of the third parameter. As seen from these figures, the cost saving is not monotonic with respect to all these parameters. This is because the impact of each parameter on the cost saving is multi-fold. When the storage cost rate $u[s_x]$ increases, if s_x does not hold the master replica, the time period $\frac{w}{u[s_x]}$ for s_x to hold a local replica after receiving a transfer is shorter. Therefore, the additional storage period needed for s_x to host the master

replica is longer. As a result, the additional storage cost for s_x to host the master replica is higher. In the meantime, the number of transfers to s_x that can be eliminated by keeping the master replica at s_x is larger, so the transfer cost saved is also higher. The overall impact on the cost saving depends on the relative extent of these effects. Likewise, when the transfer cost w increases, if s_x does not hold the master replica, the time period $\frac{w}{u[s_x]}$ for s_x to hold a local replica after receiving a transfer is longer. Therefore, the additional storage period (cost) needed for s_x to host the master replica is shorter (lower). Meanwhile, the number of transfers to s_x that can be eliminated by keeping the master replica at s_x is smaller, though the cost saved for eliminating each transfer is higher. The overall influence on the cost saving is a combination of all these effects. Similarly, the impact of the request rate $\lambda[s_x]$ on the cost saving is also two-fold. The additional storage period and hence cost needed for s_x to host the master replica decrease with increasing request rate. This is because at a higher request rate, the number of transfers is larger. As a result, the aggregate time period for s_x to hold a local replica after receiving transfers is longer. Thus, the additional storage period needed to host the master replica is shorter. On the other hand, with increasing request rate, the number of transfers eliminated and hence the transfer cost saved per time unit in the additional storage period increases. The overall impact on the cost saving is an integration of the above effects. Somewhat surprisingly, Figure 5.1f shows that given the request rate $\lambda[s_x]$ and the transfer cost w , the cost saving does not necessarily increase with decreasing storage cost rate $u[s_x]$. This implies that the site with the lowest storage cost rate is not always the best site to host the master replica even when all sites have the same request rates.

The above analysis shows that the cost savings for keeping the master replica at different sites can be very diverse. To minimize the total cost of all sites, we should choose the site which has the largest cost saving to keep the master replica. Motivated by the above analysis, we now extend the basic online algorithm to allow the master replica to be dynamically moved among sites based on the request patterns observed. We refer to the new algorithm as the dynamic online algorithm.

Algorithm 2 shows the detailed algorithm executed by a site s_x . If s_x is not hosting the master replica, it operates as in the basic online algorithm. That is, on a request arrival, if s_x has a replica available, the request is served locally (line 38). Otherwise, a transfer is carried out from the master replica to serve the request, after which s_x will create and hold a replica for a period of $\frac{w}{u[s_x]}$ (lines 16–17 and 37).

The key idea of the dynamic online algorithm is to choose the site that can save the cost most for keeping the master replica. To do so, we need to estimate the cost savings for the sites. This is done by the site hosting the master replica. When a site s_x starts holding the master replica at time $\mathbf{t}_{\text{start}}$, it initializes the accumulated cost saving $\text{saving}[\mathbf{z}] = 0$ and the amortized cost saving $\text{amsaving}[\mathbf{z}] = 0$ for each site s_z , and a counter $\text{counter}[\mathbf{z}] = 0$ for each site s_z to record the number of outgoing transfers to s_z (line 9). The cost savings are updated incrementally when s_x performs new outgoing transfers to other sites. Suppose that a new transfer is conducted at time $\mathbf{t}$ to a site s_y due to a request arriving at s_y . When asking for the transfer, site s_y also informs s_x about the time $\mathbf{t}_p$ of its previous received request (lines 15 and 40). Then, the cost incurred between $\mathbf{t}_p$ and $\mathbf{t}$ is $w + u[s_y] \cdot (\mathbf{t}_p + \frac{w}{u[s_y]} - \mathbf{t}_p)$. On the other hand, if the master replica were hosted by s_y , the storage cost

Algorithm 2 Dynamic online algorithm executed by a site s_x

```

1: Initialization (at time  $t_0$ ):
2:    $t_{\text{expire}} \leftarrow t_0$ ; //expiration time of local replica
3:   if  $s_x = s_{\text{min}}$  then
4:     create local replica and keep it as master replica;
5:      $\text{master} \leftarrow s_{\text{min}}$ ; //where master replica is located
6:      $t_{\text{start}} \leftarrow t_0$ ; //when  $s_x$  starts holding master replica
7:      $t_{\text{plocal}} \leftarrow t_{\text{start}}$ ; //time of last update of  $\text{saving}[x]$ 
8:      $\text{totalcount} \leftarrow 0$ ; //number of outgoing transfers
9:     initialize  $\text{saving}[z] \leftarrow 0$ ,  $\text{amsaving}[z] \leftarrow 0$ ,  $\text{counter}[z] \leftarrow 0$  for each site  $s_z$ ;
10:    notify all other sites of  $\text{master}$ ;
11:  end if
12: On receiving a request at time  $t$ :
13:  if  $s_x \neq \text{master}$  then
14:    if  $t > t_{\text{expire}}$  then
15:      inform site  $\text{master}$  of  $t_p$  (the arrival time of last local request at  $s_x$ );
16:      ask for transferring object from site  $\text{master}$ ;
17:      create local replica;
18:      if transfer has “move” tag then
19:        keep local replica as master replica;
20:         $\text{master} \leftarrow s_x$ ,  $t_{\text{start}} \leftarrow t$ ,  $t_{\text{plocal}} \leftarrow t_{\text{start}}$ ;
21:        execute lines 8–10;
22:      end if
23:    end if
24:  else
25:    if  $t > t_{\text{expire}}$  then
26:       $\text{saving}[x] \leftarrow \text{saving}[x] + w$ ;
27:    end if
28:     $\text{saving}[x] \leftarrow \text{saving}[x] - u[s_x] \cdot \max\{0, t - \max\{t_{\text{plocal}}, t_{\text{expire}}, t_{\text{start}}\}\}$ ;
29:     $\text{amsaving}[x] \leftarrow \text{saving}[x] / (t - t_{\text{start}})$ ;
30:     $t_{\text{plocal}} \leftarrow t$ ;
31:    if  $\text{amsaving}[x] = \max_{z=1}^m \{\text{amsaving}[z]\}$  and  $\text{totalcount} \geq \text{threshold}$  then
32:      continue local replica as master replica;
33:       $t_{\text{start}} \leftarrow t$ ,  $t_{\text{plocal}} \leftarrow t_{\text{start}}$ ;
34:      execute lines 8–10;
35:    end if
36:  end if
37:   $t_{\text{expire}} \leftarrow t + w/u[s_y]$ ;
38:  serve the request with local replica;
39: When another site  $s_y$  asks for transferring object from  $s_x$  at time  $t$ : // this implies  $s_x = \text{master}$ 
40:  receive  $t_p$  from site  $s_y$ ;
41:   $\text{saving}[y] \leftarrow \text{saving}[y] + w - u[s_y] \cdot (t - \max\{t_p + \frac{w}{u[s_y]}, t_{\text{start}}\})$ ;
42:   $\text{amsaving}[y] \leftarrow \text{saving}[y] / (t - t_{\text{start}})$ ;
43:   $\text{counter}[y] \leftarrow \text{counter}[y] + 1$ ;
44:   $\text{totalcount} \leftarrow \text{totalcount} + 1$ ;
45:  execute lines 28–35;
46:  if  $\text{amsaving}[y] = \max_{z=1}^m \{\text{amsaving}[z]\}$  and  $\text{counter}[y] \geq 2$  and  $\text{totalcount} \geq \text{threshold}$  then
47:    add “move” tag to the transfer to site  $s_y$ ;
48:     $\text{master} \leftarrow s_y$ ;
49:     $t_{\text{expire}} \leftarrow t_h + \frac{w}{u[s_x]}$  (where  $t_h$  is the arrival time of last local request at  $s_x$ );
50:  end if
51:  transfer object to site  $s_y$ ;
52: On expiration of local replica at time  $t_{\text{expire}}$ :
53:  delete local replica;

```

incurred is $u[s_y] \cdot (t - t_p)$ and no transfer cost is incurred. Thus, the cost saving for keeping the master replica at s_y is $w - u[s_y] \cdot (t - (t_p + \frac{w}{u[s_y]}))$. Since we are interested in only the cost saving after s_x takes up the master replica, we further take the larger value between $t_p + \frac{w}{u[s_y]}$ and t_{start} , and increase the cost saving $\text{saving}[y]$ by $w - u[s_y] \cdot (t - \max\{t_p + \frac{w}{u[s_y]}, t_{\text{start}}\})$ (line 41). s_x computes the amortized cost saving for site s_y since t_{start} as $\text{saving}[y]/(t - t_{\text{start}})$ (line 42). After the updates, if site s_y has the highest amortized cost saving among all sites (line 46), s_x moves the master replica to s_y by leveraging the transfer to s_y . That is, s_x adds a “move” tag to the transfer to signal s_y (line 47). When s_y sees the “move” tag, it creates a replica and keeps it as the master replica (line 19). Then, s_y notifies all sites about the new master replica, and initializes $\text{saving}[z]$, $\text{amsaving}[z]$, and $\text{counter}[z]$ for each site s_z (line 21).

To collect sufficient statistics for stable estimation of cost savings, s_x considers moving the master replica to another site only if it has conducted at least a **threshold** number of outgoing transfers since it started holding the master replica (line 46). An additional condition for s_x to move the master replica to another site s_y is that s_x must have conducted at least two outgoing transfers to s_y , i.e., $\text{counter}[y] \geq 2$ (line 46). This ensures that s_x has observed at least one complete inter-transfer interval for s_y , so that the cost saving statistics are more reliable. If s_x has only one outgoing transfer to a site s_y out of a **threshold** number of outgoing transfers, it may be because either (1) the request arrival rate at s_y is rather high so that it constantly keeps a local replica even if it is not designated to hold the master replica; or (2) the request arrival rate at s_y is quite low so that it asks for transfers from the master replica fairly infrequently. In either case, there is no need

to move the master replica to s_y .

Suppose that the last local request at s_x before moving the master replica to another site s_y is at time t_h . Then after moving the master replica to s_y , s_x sets the expiration time of its local replica to $t_h + \frac{w}{u[s_x]}$ (line 49). In this way, the set of s_x 's local requests that will lead to incoming transfers afterwards would be the same as if s_x had not held the master replica. During the time period when s_x is holding the master replica, it also keeps track of which local requests would result in (hypothetical) incoming transfers if it were not holding the master replica. Then s_x can also estimate its own amortized cost saving and include it in the comparison. Since s_x has the time information of all local request arrivals, it can update its cost saving statistics more frequently for higher accuracy. Specifically, s_x updates its cost saving **saving**[x] at every local request arrival (lines 25–28) and every outgoing transfer to other sites (line 45). To do so, s_x keeps track of the time t_{local} when **saving**[x] was last updated. When a new local request arrival or outgoing transfer occurs at time t , **saving**[x] is decreased by $u[s_x] \cdot \max\{0, t - \max\{t_{\text{local}}, t_{\text{expire}}, t_{\text{start}}\}\}$ (line 28), which is the additional cost for keeping the master replica over the time period in (t_{local}, t) that is (1) after t_{expire} (not covered by the local replica if the master replica is not hosted locally) and (2) after t_{start} (after s_x takes up the master replica). When a local request arrival results in a (hypothetical) incoming transfer, **saving**[x] is increased by the transfer cost w (lines 25–27). In either case, the amortized cost saving for s_x is updated as **saving**[x]/($t - t_{\text{start}}$) (line 29). After the updates, if site s_x has the highest amortized cost saving among all sites (line 31), it continues holding the master replica and resets **saving**[z], **amsaving**[z], and **counter**[z] for each site s_z (lines 32–34).

The dynamic online algorithm is lightweight. The time complexity of the algorithm executed by the site holding the master replica on receiving each local request or conducting each outgoing transfer is $O(m)$, where m is the number of sites. If the amortized cost savings `amsaving[z]` are maintained using the heap structure, the time complexity can be further reduced to $O(\log m)$. Since each site have a space complexity of $O(m)$ for storing arrays `saving[]`, `amsaving[]` and `counter[]`, the space complexity of the dynamic online algorithm is $O(m^2)$.

Chapter 6

Experimental Evaluation

6.1 Experimental Setup

We conduct simulation experiments to evaluate the proposed algorithms by using the Google Cloud Storage pricing data [1]. We select three sites in Europe: Finland, London, and Zurich. Detailed prices of these sites are listed in Table 6.1.

Table 6.1: Google cloud storage pricing [1]

Site	Finland	London	Zurich
Storage price (per GB per time unit)	0.020	0.023	0.025
Transfer price (per GB)	0.010	0.010	0.010

We assume that request arrivals at each site follow a Poisson process. The request rates to different sites follow a Zipf distribution. Specifically, the sites are arranged into a list. The request rate to the i -th site in the list ($i = 1, 2, 3$) is set to $\alpha \cdot i^{-\beta} / \sum_{j=1}^3 j^{-\beta}$, where α is the total rate of requests and $\beta \geq 0$ is the Zipf parameter. The larger the value of β , the more skewed the request distribution. When $\beta = 0$, all sites share the same request rates. We test three settings: ASCEND, DESCEND and FLUCTUATE.

In the ASCEND (DESCEND) setting, the sites are listed in the ascending (descending) order of storage prices so that the lowest-priced (highest-priced) site receives the highest volume of requests. In the FLUCTUATE setting, the sites are listed in the order of [London, Finland, Zurich], where the London site has the highest request rate. As a result, the storage prices of the sites are not monotonic with respect to the request volumes.

We implement the optimal offline and dynamic online algorithms described in Sections 4.2 and 5.2 respectively. In the dynamic online algorithm, the **threshold** is set to 30. In addition, we also implement four baseline algorithms. In the first baseline called Baseline(All Sites), all the sites constantly hold local replicas so that no transfer is needed to serve any request. In the other three baselines, the master replica is always kept at a fixed site. These three baselines are referred to as Baseline(Finland), Baseline(London), and Baseline(Zurich). Note that Baseline(Finland) is actually the basic online algorithm described in Section 5.1 since the Finland site has the lowest storage price. The objective of the experiments is to provide an overview of the relative performance of different algorithms. To this end, we conduct experiments over a wide range of parameter settings to study their impacts. Each experiment run simulates a period of 200 time units. We compute the total storage and transfer cost produced by each algorithm for comparison. The size of the data object is assumed to be 1 GB.

6.2 Experimental Results

Figures 6.1, 6.2 and 6.3 show the cost for different request rates and Zipf parameters. We make the following observations:

First, the cost of the basic online algorithm (i.e., Baseline(Finland)) nor-

Figure 6.1: Cost for ASCEND setting

Figure 6.2: Cost for DESCEND setting

Figure 6.3: Cost for FLUCTUATE setting

Figure 6.4: Cost ratio between basic online and optimal offline algorithms

(a) Cost ratio for ASCEND setting

(b) Cost ratio for DESCEND setting

(c) Cost ratio for FLUCTUATE setting

Figure 6.5: Cost ratio between dynamic online and optimal offline algorithms

malized by that of the optimal offline algorithm is typically much lower than the theoretical bound 3. In fact, the cost ratio between the basic online algorithm and the optimal offline algorithm is below 2 for all the cases tested (see Figure 6.4). The dynamic online algorithm outperforms the basic online algorithm. The cost ratio between the dynamic online algorithm and the optimal offline algorithm is below 1.6 for all the cases tested (see Figure 6.5), indicating that the performance of the dynamic algorithm is close to optimal. This shows the importance of dynamic replication for cost optimization.

Second, when the request rate is not high, Baseline(All Sites) has a much higher cost than the dynamic online algorithm and the other three baselines which keep only one master replica. This is because when the request rate is not high, it is more cost-effective to store replicas at fewer sites and use

transfers to serve requests at different sites. On the other hand, when the request rate is high, by Proposition 2, the optimal replication strategy is to constantly keep a replica at every site and eliminate all transfers. In this case, Baseline(All Sites) performs close to the optimum. We also observe that for a certain range of request rate, Baseline(All Sites) performs slightly better than the other three baselines and the dynamic online algorithm. This is because in the latter four algorithms, a site s_x not holding the master replica would keep a local replica for only a period of $\frac{w}{u[s_x]}$ after receiving a transfer from the master replica. If the next request arrives within this period, the cost incurred to serve the next request will include only the storage cost and is optimal. If the next request arrives beyond this period, the total storage and transfer cost incurred to serve the next request will be $2w$. In the case that the next request arrives between $\frac{w}{u[s_x]}$ and $2\frac{w}{u[s_x]}$ after the current request, this total cost is in fact larger than the cost of holding the local replica till the next request to serve it (which is between w and $2w$). Thus, in this case, Baseline(All Sites), by constantly keeping a replica at every site, performs slightly better.

Third, the relative performance of the three baselines Baseline(Finland), Baseline(London) and Baseline(Zurich) varies widely. As discussed in Section 5.2, the cost saving for keeping the master replica at a site depends on the request rate to the site, the storage cost of the site and the transfer cost. Figures 6.1b–6.1d show that even if the request rates are higher at lower-priced storage sites (the ASCEND setting), the site with the lowest storage price (Finland) may not always be the best site to host the master replica. Figures 6.2b–6.2d show that in the DESCEND setting, the site with the lowest storage price (Finland) is often not the best site to host the

(a) Running time for ASCEND setting

(b) Running time for DESCEND setting

(c) Running time for FLUCTU-ATE setting

Figure 6.6: Running time for optimal offline algorithm

master replica. Neither is the site with the highest request rate (Zurich) always the best site to host the master replica. In general, the cost difference among the three baselines Baseline(Finland), Baseline(London) and Baseline(Zurich) increases with the skewness of request distribution (i.e., β). For almost all the settings tested, the dynamic online algorithm produces a cost close to the best (lowest) among Baseline(Finland), Baseline(London) and Baseline(Zurich). This indicates that the dynamic online algorithm is able to move the master replica to the best site for saving cost.

Figures 6.6 and 6.7 show the running time of our optimal offline and dynamic online algorithms. In all our experiments, it takes less than 2 seconds to run the optimal offline algorithm and 0.5 second to run the dynamic online algorithm on a machine with Intel Core 2.80GHz CPU and 16GB memory.

(a) Running time for ASCEND setting

(b) Running time for DESCEND setting

(c) Running time for FLUCTU-ATE setting

Figure 6.7: Running time for dynamic online algorithm

This demonstrates the efficiency of our algorithms. For both algorithms, the running time increases with the request rate due to more requests to process.

Chapter 7

Conclusion

In this thesis, we have investigated the cost optimization problem for distributed cloud storage under a heterogeneous billing model. Both the offline and online settings are studied. In the offline setting, we have developed an optimal algorithm given the information of the full request sequence. In the online setting where no information about future requests is available, we have proposed two distributed algorithms. We have performed extensive experiments and shown that the cost incurred by the proposed dynamic online algorithm is close to the optimal offline algorithm over a wide range of parameter settings. One possible direction for future work is to extend the algorithms to deal with potentially different transfer costs between various sites.

References

- [1] “Google cloud storage pricing,” <https://cloud.google.com/storage/pricing>.
- [2] K. Kalpakis, K. Dasgupta, and O. Wolfson, “Optimal placement of replicas in trees with read, write, and storage costs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 628–637, 2001.
- [3] X. Tang and J. Xu, “QoS-aware replica placement for content distribution,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 10, pp. 921–932, 2005.
- [4] S. Agarwala, D. Jadav, and L. A. Bathen, “iCostale: adaptive cost optimization for storage clouds,” in *IEEE International Conference on Cloud Computing*, 2011, pp. 436–443.
- [5] U. Bellur, A. Malani, and N. C. Narendra, “Cost optimization in multi-site multi-cloud environments with multiple pricing schemes,” in *IEEE International Conference on Cloud Computing*, 2014, pp. 689–696.
- [6] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, “CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster,” in *IEEE International Conference on Cluster Computing*, 2010, pp. 188–196.

- [7] N. K. Gill and S. Singh, “A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers,” *Future Generation Computer Systems*, vol. 65, pp. 10–32, 2016.
- [8] T. G. Papaioannou, N. Bonvin, and K. Aberer, “Scalia: An adaptive scheme for efficient multi-cloud storage,” in *International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–10.
- [9] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, “Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services,” in *ACM SOSP*, 2013, pp. 292–308.
- [10] S.-Q. Long, Y.-L. Zhao, and W. Chen, “Morm: A multi-objective optimized replication management strategy for cloud storage cluster,” *Journal of Systems Architecture*, vol. 60, no. 2, pp. 234–244, 2014.
- [11] Y. Mansouri and A. Erradi, “Cost optimization algorithms for hot and cool tiers cloud storage services,” in *IEEE International Conference on Cloud Computing*, 2018, pp. 622–629.
- [12] D. Yuan, Y. Yang, X. Liu, W. Li, L. Cui, M. Xu, and J. Chen, “A highly practical approach toward achieving minimum data sets storage cost in the cloud,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1234–1244, 2013.
- [13] A. Khanafer, M. Kodialam, and K. P. N. Puttaswamy, “The constrained ski-rental problem and its application to online cloud cost optimization,” in *IEEE INFOCOM*, 2013, pp. 1492–1500.

- [14] N. L. Scouarnec, C. Neumann, and G. Straub, “Caching policies for cloud-based systems: To keep or not to keep,” in *IEEE International Conference on Cloud Computing*, 2014, pp. 1–8.
- [15] B. Veeravalli, “Network caching strategies for a shared data distribution for a predefined service demand sequence,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1487–1497, 2003.
- [16] Y. Wang, B. Veeravalli, and C.-K. Tham, “On data staging algorithms for shared data accesses in clouds,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 4, pp. 825–838, 2013.
- [17] Y. Wang, S. He, X. Fan, C. Xu, and X.-H. Sun, “On cost-driven collaborative data caching: A new model approach,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 662–676, 2018.
- [18] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, “Competitive snoop caching,” *Algorithmica*, vol. 3, no. 1-4, pp. 79–119, 1988.