Economical Aspects of Resource Allocation under Discounts

by

Xinhui Hu

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved March 2015 by the
Graduate Supervisory Committee:

Andrea Richa, Chair
Stefan Schmid
Arunabha Sen
Guoliang Xue

ARIZONA STATE UNIVERSITY

May 2015

ABSTRACT

Resource allocation is one of the most challenging issues policy decision makers must address. The objective of this thesis is to explore the resource allocation from an economical perspective, i.e., how to purchase resources in order to satisfy customers' requests. This thesis attends to answer the question: when and how to buy resources to fulfill customers' demands with minimum costs?

The first topic studied in this thesis is resource allocation in cloud networks. Cloud computing heralded an era where resources (such as computation and storage) can be scaled up and down elastically and on demand. This flexibility is attractive for its cost effectiveness: the cloud resource price depends on the actual utilization over time. This thesis studies two critical problems in cloud networks, focusing on the economical aspects of the resource allocation in the cloud/virtual networks, and proposes six algorithms to address the resource allocation problems for different discount models. The first problem attends a scenario where the virtual network provider offers different contracts to the service provider. Four algorithms for resource contract migration are proposed under two pricing models: Pay-as-You-Come and Pay-as-You-Go. The second problem explores a scenario where a cloud provider offers k contracts each with a duration and a rate respectively and a customer buys these contracts in order to satisfy its resource demand. This work shows that this problem can be seen as a 2-dimensional generalization of the classic online parking permit problem, and present a k-competitive online algorithm and an optimal offline algorithm.

The second topic studied in this thesis is to explore how resource allocation and purchasing strategies work in our daily life. For example, *is it worth buying a Yoga pass which costs USD 100 for ten entries, although it will expire at the end of this year?* Decisions like these are part of our daily life, yet, not much is known today about good online strategies to buy discount vouchers with expiration dates. This work hence

introduces a Discount Voucher Purchase Problem (DVPP). It aims to optimize the strategies for buying discount vouchers, i.e., coupons, vouchers, groupons which are valid only during a certain time period. The DVPP comes in three flavors: (1) Once-Expired-Lose-Everything (OELE): Vouchers lose their entire value after expiration. (2) Once-Expired-Lose-Discount (OELD): Vouchers lose their discount value after expiration. (3) Limited Purchasing Window (LPW):Vouchers have the property of OELE and can only be bought during a certain time window.

This work explores online algorithms with a provable competitive ratio against a clairvoyant offline algorithm, even in the worst case. In particular, this work makes the following contributions: this work presents a 4-competitive algorithm for OELE, an 8-competitive algorithm for OELD, and a lower bound for LPW. This work also presents an optimal offline algorithm for OELE and LPW, and shows it is a 2-approximation solution for OELD.

DEDICATION

I dedicate my dissertation work to my loving parents, Cunbiao Hu and Liqin Wu, for making me be who I am!

I also dedicate this dissertation to my precious husband, Xi Fang, for supporting me all the way! Without his help and encouragement it simply never would have been.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincerest gratitude to my advisor, Dr. Andrea Richa, for her professional guidance, patience, support, and encouragement over my PhD study. Her immense knowledge and enthusiastic attitude towards research quality have always inspired me. I greatly appreciate her invaluable guidance in my research and her warm-hearted help in my personal life. I am very fortunate to have her as my advisor without whom I could not have my current achievement.

I would also like to thank all my committee members, Dr. Guoliang Xue, Dr. Arunahba Sen, and Dr. Stefan Schmid for their invaluable advice. I learned a lot from my committee members about research either by doing research with them or taking their classes.

I also thank my colleagues and friends: Dr. Jin Zhang, Mengxue Liu, Chenyang Zhou, Zahra Derakhshandeh, Dr. Dejun Yang, Dr. Lingjun Li, Xinxin Zhao, Xiang Zhang, Ruozhou Yu and Dr. Ziming Zhao for the pleasant and inspiring discussion. Their friendship is a valuable experience for me.

Last but not least, I would like to thank my mother Liqin Wu, my father Cunbiao Hu, and my husband Xi Fang, who give me endless love and constant encouragement during my study at ASU and throughout my whole life. I would like to thank my daughter. She is the best present for me. This dissertation is dedicated to them!

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Allocation of network resources is one of the most challenging and critical issues that network administrators must carefully address. This thesis attends to this issue by addressing economical aspects of resource allocation , namely we consider in broad terms the problem of when and how to buy resources to fulfill customers' demands with minimum cost.

The ***first*** research topic studied in this thesis is resource allocation in cloud networks. Nowadays, the Internet is an integrated part of our daily life and information spread becomes extremely fast. An emerging technology, called network virtualization, helps realize the vision of an Internet where resources offered by different stakeholders are used and shared by multiple co-existing virtual networks. The abstraction introduced by network virtualization opens many new business opportunities. It is expected that in the near future, infrastructure providers, resource brokers, and resource resellers will offer flexibly specifiable and on-demand virtual networks over the Internet, similarly to the traditional elastic resources provided by today's clouds. This motivates us to explore virtual network resource allocation problems in clouds from an economical perspective.

Resources in clouds (e.g., CPUs and storages) are provided as general utilities that can be leased and released by users in an on-demand fashion through the Internet. Network virtualization not only introduces an Internet-wide resource market, but also initiates many new business models. One example is that a startup company running web services no longer needs to invest in its own infrastructure, but can dynamically lease cloud resources to provide the services to its customers in a dynamic and cost-

efficient manner. For instance, online content provider Netflix is able to support seamless global service by partnering with Amazon Web Services (AWS) for services and delivery of content. AWS enables Netflix to quickly deploy thousands of servers and terabytes of storage within minutes. Users can stream Netflix shows and movies from anywhere in the world, including on the web, on tablets, or on mobile devices such as iPhones. Another example is that of a hybrid resource provision framework. Specifically, virtualized cloud resources can be used to help an infrastructure provider deal with a large amount of loads during peak periods when this provider does not have enough capacity in its infrastructure. Resource brokerage is another emerging business model. A broker may lease a large amount of network resources from different providers and resell them to its customers (at a higher price).

Most of the previous works for resource allocation focus on migration or resource embedding in the networks. In this thesis, we attend to the economical aspects of the resource allocation, i.e., cost reduction. In the perspective of the service providers, they are faced with the challenge that while moving the server closer to the customers improves QoS, frequent migrations come at service interruption and bulk data transfer costs. We aim to minimize the total cost by balancing the tradeoff among all types of the costs. In the perspective of the service customers, they are faced with the challenge that its resource demand is not known in advance. In order to satisfy its demand at any time, and to avoid over-purchasing of the service, resource contracts need to be bought in advance. The goal is an online purchasing policy in which the gap between the costs of the bought contracts and the costs of the actual needed contracts is small.

Therefore, this thesis studies two critical problems in cloud networks, focusing on the economical aspects of the resource allocation in the cloud/virtual networks The first problem attends a scenario where the virtual network provider offers different

contracts to the service provider. Four algorithms for resource contract migration are proposed under two pricing models: Pay-as-You-Come and Pay-as-You-Go. The second problem explores a scenario where a cloud provider offers k contracts each with a duration and a rate respectively and a customer buys these contracts in order to satisfy its resource demand. We show that this problem can be seen as a 2-dimensional generalization of the classic online parking permit problem, and present a k-competitive online algorithm and an optimal offline algorithm.

The ***second*** research topic studied in this thesis is to explore how resource allocation/purchasing strategies work in our daily life. As the Internet becomes widespread and the information spreads quickly, the competition among different companies becomes more and more drastic. In order to attract more customers in such a highly competitive market, some companies or even brokers offer discount vouchers to improve the profits. For example, a company called Groupon, offers one Groupon voucher per day in each of the markets it serves where each Groupon may have one or more discount rates.

Naturally, from customers' perspective, they aim to minimize the overall money spent and only buy the vouchers or passes if these are really needed. Let us consider an example scenario, where Alice regularly takes yoga classes, and wants to decide whether she shall buy a yoga pass for the next year (the pass will expire at the end of the year). The yoga pass costs USD C, and allows Alice to join $\frac{C}{\pi(p)}$ yoga classes each at a discounted price $\pi(p)$ instead of the original price p per class. Unfortunately, however, Alice is not sure on how long she will be interested in yoga, and in the worst case, she may not attend any classes in the future after having bought the pass. This thesis studies online strategies that help Alice to decide whether it is worth buying the pass. More specifically, we are interested in online purchasing strategies which ensure that the money spent by Alice is not far from the money spent by an optimal

and clairvoyant offline algorithm which knows whether Alice will be interested in yoga in the future.

We hence introduce a discount purchase problem, called the Discount Voucher Purchase Problem (DVPP). It aims to optimize the strategies for buying discount vouchers, i.e., coupons, vouchers, groupons which are valid only during a certain time period. The DVPP comes in three policies:

Once expire lose everything (OELE): The voucher loses its entire value after its expiration date.

Once expire lose discount (OELD): The voucher loses its discount for the unused face value after its expiration date.

Limited purchasing window (LPW): The voucher not only has the same property as that of OELD, but also has a limited purchasing window allowed for purchasing by customers.

We present an online algorithm that can compute a 4-competitive solution to OELE and an 8-competitive solution to OELD. We also prove a lower bound on online competitive ratio for LPW. In addition, we present an offline algorithm that can compute an optimal solution to OELE and LPW and a 2-competitive solution to OELD.

This thesis is organized as follows: In Chapter 1 we define the research problems studied in this thesis. In Chapter 2, we study two pricing strategies Pay-as-You-Come and Pay-as-You-Go for the optimal service migration in virtual networks, and then discuss an online algorithm. In Chapter 3, we study an online cloud resource allocation problem and analyze both upper and lower bounds on the competitive ratios. An optimal offline algorithm is proposed as a blackbox for this online algorithm. In Chapter 4, we study the Discount Voucher Purchase Problem (DVPP) and discuss

three variants of DVPP. We show competitive online algorithms for OELE and OELD and discuss the lower bound for LPW. We present an offline algorithm for OELE and LPW, and show it is 2-approximation to OELD. In Chapter 5 we conclude the completed work and discuss lines for future work.

Chapter 2

OPTIMAL MIGRATION CONTRACTS IN VIRTUAL NETWORKS: PAY-AS-YOU-COME VS PAY-AS-YOU-GO PRICING

2.1 Introduction

The Internet becomes more and more virtualized and programmable (or "software-defined"), and we witness a trend towards extending the cloud paradigm to the *network*. Researchers in the field of *network virtualization* develop prototype architectures that herald flexibly specifiable, fully *virtual networks* (VNets) (also known as *CloudNets*): virtual networks that can be requested at short notice (and even be migrated arbitrarily within the specification constraints), while providing isolation guarantees (e.g., in terms of QoS or security). This paradigm has the potential to open a network infrastructure for a wide range of new and innovative services, and it is believed that new economical entities will emerge that lease (or re-lease) infrastructure parts to service providers.

We expect that in the near future, such virtual networks connecting arbitrary locations (and spanning multiple autonomous systems and providers) in the Internet can be leased similarly to the resource leasing models of today's clouds. The work in this chapter attends to a use case for such dynamic VNets where a service provider offers a flexible and latency-critical service (for instance a web service, an SAP server or a game server) to its mobile customers whose demand and locations changes over time (e.g., due to time-zone effects or commuting). We assume that the service provider itself uses the resource services of a substrate infrastructure provider (e.g., a physical infrastructure provider or a virtual network provider) in order to offer a low-

latency access to a server which can be migrated seamlessly in the VNet (i.e., without reconfiguration or changes of routable network addresses). The service provider is faced with the challenge that while moving the server closer to the customers improves QoS (and/or reduces roaming costs), frequent migrations come at service interruption and bulk data transfer costs. We initiate the study of optimal offline and online migration strategies for the service provider under two different pricing models.

Our Contribution. This work initiates the study of the virtual server migration problem from an economical perspective. We compare the two most basic pricing policies *Pay-as-You-Come* and *Pay-as-You-Go* (see, e.g., [34]), in which a service provider has to pay in advance for time-based contracts respectively in retrospect for the resources actually used. The service provider receives a discount when buying larger contracts, e.g., a contract of twice the resource volume only costs 50% more. As a first step, we design offline migration algorithms for different settings and discount functions. We find that optimal offline solutions can indeed be computed in polynomial time by using non-trivial dynamic programs. This work also initiates exploring online migration strategies.

Work Organization. The rest of this chapter is organized as follows. We discuss related work in section 2.2. In section 2.3, we discuss the model used in this work. We present two offline migration strategies in section 2.4. As a first look, we have a short discussion on online algorithm for both pricing strategies in section 2.6. We conclude this problem in section 2.7.

2.2 Related Work

Our work is motivated by the advent of first network virtualization prototype architectures such as GENI. For a good overview of the network virtualization field, see [14]. Theoretical research on network virtualization often focuses on the problem

of how to *embed* VNets, e.g., [13, 48, 37] (and especially the survey [8]), while benefitting from specification flexibilities [29]. Naturally, there are also many papers and results on migration (e.g., [2, 5, 23, 45]): the possibility to migrate is one of the key advantages of the virtualization abstraction; it is due to the decoupling of services from the physical infrastructure. Indeed, it has been shown that it can make sense to migrate a Samba front-end server closer to the clients even for bulk-data applications [24]. Our work builds upon the formal migration model studied in [5] and ports it to an economical setting.

Economical aspects of network virtualization are much less well-understood, but there exist strong ties with related problems in, e.g., cloud computing. For example, Armbrust et al. [30] made an effort to understand cloud computing economical models for long-term hosting a service in the cloud. Dash et al. [17] proposed an economic model for self-tuned cloud caching targeting the service of scientific data. Recently, Pal and Hui [34] devised and analyzed three inter-organizational economic models relevant to cloud networks, and formulated non-cooperative price and QoS games between multiple cloud providers existing in a cloud market. In the context of network virtualization, Schaffrath et al. [38] identified stakeholders and economical roles in a network virtualization environment. The authors distinguish between a physical infrastructure provider, a virtual network provider (i.e., resource reseller), a virtual network operator and a service provider. In terms of pricing, Even et al. [18] presented an online algorithm which decides which VNets to accept and embed such that the overall provider *benefit* is maximized. The benefit threshold of when to accept a VNet can be seen as a simple form of pricing. Migration is not considered in [18].

Finally, a description of our own network virtualization prototype (currently using VLANs) which is developed at Telekom Innovation Laboratories and NTT DoCoMo Eurolabs and which motivates our work can be found in [38]. Currently, migration

is seamless (i.e., without the need for reconfigurations) but not live. See [16] for a migration demo.

2.3 Model

A virtual network topology can be modeled as a graph $G = (V, E)$ where $V(G)$ denotes the set of nodes and $E(G)$ the set of links. We assume that a service provider can place its service (i.e., the server) on any location in the virtual network. Requests can originate from different access points in $V(G)$, and the access cost is given by the shortest path (depending on some given metric D) to the server location in $V(G)$. In order to reduce the access cost, the virtual server can be migrated along the links in $E(G)$. To do so, the service provider needs to purchase bandwidth along the migration path.

We attend to a scenario where a virtual network provider offers the service provider a choice of contracts of different durations in which dedicated resources can be leased in the virtual network (e.g., for migration), i.e., $\mathfrak{D} = \{d_1, d_2, \ldots, d_k\}$ (we assume $d_1 < d_2 < \ldots < d_k$). In addition to the contract durations, the service provider can choose between different bandwidths along the links, i.e., it can choose among the following set of bandwidths for each link: $\mathfrak{B} = \{b_1, b_2, \ldots, b_q\}$ (we also assume that $b_1 < b_2 < \ldots < b_q$).

We consider two different pricing models. Under *Pay-as-You-Go* pricing, a customer only needs to pay for the used resources after the actual consumption (or at regular time intervals T), and the best contract is determined according to the usage pattern *a posteriori*. Pay-as-You-Go pricing is often used in the context of cloud resource leasing. In contrast, in the *Pay-as-You-Come* model, a customer needs to decide *in advance* which kind of *time-based* contracts she is interested in, and needs to buy them before the actual resource usage. Examples for this model can be found,

e.g., in the context of private Internet access where users often pay in advance and independently of the actual usage pattern.

In this work, in order to focus on the main tradeoffs, we initiate the study of these pricing models in a simplified scenario where the virtual network consists of two locations only (e.g., one in the U.S. and one in Asia); we will refer to these locations by L (left) and R (right) respectively and normalize their distance to one unit. The server can be migrated arbitrarily between the two locations if a corresponding resource contract is present for the bulk-data transfers. Concretely, a contract in the *Pay-as-You-Come* model consists of a duration d_i and the bandwidth b_j to lease the virtual link between the two sites for d_i units (e.g., days) and at a bandwidth of b_j (e.g., Mbit/s). The price of the contract is given by a function $f(d_i, b_j)$, where $f(\cdot, \cdot)$ describes a monotonic increasing discount over the contract duration and over the amount of reserved resources. For example, a twice as long contract may cost only 50% more, and doubling the reserved bandwidth may cost only 30% more. In the *Pay-as-You-Go* model, the customer only needs to pay when the service is finished or after a given duration, i.e., every T time units (e.g., a month), and only for the resources (and bulk data transfers) that are actually used. Concretely, if μ_i migrations are performed during the time period T at a bandwidth of $b_j \in \mathfrak{B}$, the overall costs amount to $f(\mu_i, b_j)$.

The main objective is to minimize the migration and contracting costs (denoted by MigCost and ConCost) while providing QoS guarantees (minimize access cost AccCost). Hence, we seek to minimize the following cost function:

$$\text{Cost} = \text{AccCost} + \text{MigCost} + \text{ConCost}$$

We assume there are n requests total, denoted by a set $< r_1, r_2, \cdots, r_n >$ at respective times $< t_1, t_2, \cdots, t_n >$. The access cost is given by the latency of the re-

quests $r_i \in V(G)$ to the location of the server $s_i \in V(G)$, i.e., AccCost $= \sum_i D[r_i, s_i]$ where r_i and s_i denote the ith request node and the server location at time t_i. The migration cost MigCost is given by the service interruption time (see also [10]), i.e., the time to transfer the server which is determined by the bandwidth of the weakest link along the migration path. (In a system supporting live migration, this cost can be negligible and set to zero.) Concretely, the migration cost is computed as MigCost $= \sum_i S \cdot D[s_{i-1}, s_i]/b_i$, where S is the server size, $D[s_{i-1}, s_i]$ denotes whether the locations $s_{i-1} \in \{L, R\}$ and $s_i \in \{L, R\}$ differ (recall that $D[s_{i-1}, s_i]$ is 1 if $s_{i-1} \neq s_i$, and 0 otherwise), and $b_i \in \mathfrak{B}$ is the (minimal) bandwidth along the migration path. Finally, the contract cost is computed as described above, i.e., ConCost $= \sum_i f(d_i, b_i)$ for the Pay-as-You-Come model and as ConCost $= f(\mu, b_i)$ for Pay-as-You-Go model, where $d_i \in \mathfrak{D},\ b_i \in \mathfrak{B}$ and μ is the total number of migrations.

The following table summarizes the formalism used in this chapter.

Terminology	
$AccCost$	Access cost
$MigCost$	Migration cost
$ConCost$	Contract cost
n	Number of requests
$r_i, 1 \leq i \leq n$	Origin of ith request
$t_i, 1 \leq i \leq n$	Time of ith request
L, R	Left node and right node
$\mathfrak{D}$	Set of k contract durations
$\mathfrak{B}$	Set of q different bandwidths
f	Discount function
S, s_{init}	Server size and initial server position
s, s'	Server location at the beginning and the end of a time step
$C_{n\times n\times 4}$, $(C_m)_{n\times n\times 4}$	Total cost matrices in PAYC, and PAYG (for each bandwidth b_m)
$(AM_m)_{n\times n\times 4}$	Combined access cost and migration cost matrix for bandwidth b_m
$(A_m)_{n\times n\times 4}$	Access cost matrix for bandwidth b_m
$(N_m)_{n\times n\times 4}$	Number of migrations matrix for bandwidth b_m

2.4 Service Migration Strategies

This section presents optimal algorithms to compute the best set of contracts and optimal migration strategies for the two presented pricing models. We will first present an algorithm PAYC for the Pay-as-You-Come model and prove its optimality, and then extend this algorithm to a PAYG algorithm which solves the Pay-as-You-Go

model. Both our algorithms PAYC and PAYG are based on dynamic programming, and fill out matrices such that optimal substructures are reused.

2.4.1 Pay-as-You-Come

Let us now turn our attention to the first, time-based pricing model. Our PAYC algorithm stores intermediate *minimum total cost* results (access, migration and contract costs) in a 3-dimensional matrix $C_{n\times n\times 4}$ where n is the total number of requests. $C[i,j,k]$ denotes an entry of the matrix, where $i,j \in [1,n]$ and $k \in \{(s,s')|s,s' \in \{L,R\}\}$. $C[i,j,(s,s')]$ denotes the minimum total cost for satisfying all requests from r_i to r_j for a scenario where at the beginning of the ith request the server is at node s and at the end of request j the server is at node s'. We also need a matrix $(AM_m)_{n\times n\times 4}$ for each bandwidth $b_m \in \mathfrak{B}$. For a fixed bandwidth b_m during the entire interval $[t_i,t_j]$, entry $AM_m[i,j(s,s')]$ stores the combined access and migration costs for the best migration strategy that satisfies the sequence of requests from r_i to r_j, assuming that the server is located at node s at the start of request r_i and at node s' at the end of request r_j. The contract costs, given by the function f, are not included in the entries of AM_m.

Given these data structures, we can describe algorithm PAYC (Algorithm 1) for the Pay-as-You-Come model. PAYC exploits that the optimal contract from request time t_i to request time t_j can either be decomposed into two consecutive subperiods with no overlapping contracts, or be obtained by buying a contract of long duration d_v and bandwidth b_m if $d_{v-1} < t_j - t_i + 1 \leq d_v$, where $d_v,\ d_{v-1} \in \mathfrak{D}$.

PAYC starts by initializing the optimal costs if we were to serve only one request r_i, for all possible combinations of starting server location s and ending server location s' at time t_i. According to our model, the access cost is equal to the distance between the current requesting node r_i and the server location s' at the end of time t_i, denoted

Algorithm 1 Algorithm PAYC

Input: Requests $<r_1, r_2, ..., r_n>$ at respective times $<t_1, t_2, ..., t_n>$.

Output: Minimum cost.

1: **for** $i = 1$ **to** n **do**

2: **for** all pairs $(s, s') \in \{L, R\}^2$ **do**

3: **for** $m = 1$ **to** q **do**

4: $AM_m[i, i, (s, s')] \leftarrow D[s', r_i] + S \cdot D[s, s']/b_m$

5: $C[i, i, (s, s')] \leftarrow \min_{1 \le m \le q}\{AM_m[i, i, (s, s')] + f(d_1 * D[s, s'], b_m)\}$

6: **for** $l = 2$ **to** n **do**

7: **for** $i = 1$ **to** $n - l + 1$ and **pairs** $(s, s') \in \{L, R\}^2$ **do**

8: $j \leftarrow i + l - 1$

9: $C[i, j, (s, s')] \leftarrow \min_{i \le u < j; s'' \in \{L,R\}}\{C[i, u, (s, s'')] + C[u + 1, j, (s'', s')]\}$

10: **if** $d_{v-1} < t_j - t_i + 1 \le d_v$, for some $v = \{1, \cdots, k\}$ **then**

11: **for** $m = 1$ **to** q **do**

12: $AM_m[i, j, (s, s')] \leftarrow \min_{s'' \in \{L,R\}}\{AM_m[i, i, (s, s'')] + AM_m[i + 1, j, (s'', s')]\}$

13: **if** $C[i, j, (s, s')] > \min_{1 \le m \le q}\{AM_m[i, j, (s, s')] + f(d_v, b_m)\}$ **then**

14: $C[i, j, (s, s')] \leftarrow \min_{1 \le m \le q}\{AM_m[i, j, (s, s')] + f(d_v, b_m)\}$

15: **return** $\min_{s_{\text{final}} \in \{L,R\}} C[1, n, (s_{\text{init}}, s_{\text{final}})]$

by $D[r_i, s']$. If the request at time t_i comes from the server location s', then no access cost is needed since $D[r_i, s']$ is 0; otherwise the access cost is positive. Recall that the migration cost for request r_i is computed as $S \cdot D[s, s']/b_m$, where $b_m \in \mathfrak{B}$ is the selected bandwidth and S is the migrated server size. We store the respective optimal cost of satisfying request r_i (which may or may not incur a non-zero access cost $D[r_i, s']$, depending on whether $r_i \neq s'$ or not) using bandwidth b_m, with starting

and ending positions of the server s and s' respectively, in $AM_m[i, i, (s, s')]$. We choose a bandwidth $b_m \in \mathfrak{B}$ such that the total cost, including the contract cost $f(d_1, b_m)$ if a migration accurs, is minimized, and store the optimal total cost in $C[i, i, (s, s')]$.

Next, we consider the total costs for sequences of more than one request. Note that there are l requests occurring between time t_i and t_j, where $i < j$ are defined in Lines 7 and 8 of the algorithm and $l\ (= j-i+1) > 1$. We have two alternative options: (i) we can split the interval $[t_i,\ t_j]$ at the time t_u of request r_u, where $i \leq u < j$, and buy contracts for the intervals $[t_i,\ t_u]$ and $[t_{u+1},\ t_j]$ independently for the two possible locations s'' of the server at time t_u; or (ii) we can buy a long contract of duration $d_v \in \mathfrak{D}$ and some bandwidth $b_m \in \mathfrak{B}$ to cover all the l requests if the period $t_j - t_i + 1$ is between d_{v-1} and d_v. The smaller cost of these two cases gives the optimal cost for the interval $[t_i,\ t_j]$.

We also update $AM_m[i, j, (s, s')]$, for all possible bandwidths b_m. Basically we extend the intervals already considered by one request (r_i), and we store in $AM_m[i, j, (s, s')]$ the migration strategy that minimizes the total access and migration costs for satisfying requests r_i through r_j using bandwidth b_m for starting and ending positions of the server s and s' respectively. Note that by taking into account all possible positions of the server at the end of request r_i, we consider all the possibilities of adding r_i to all the best possible strategies already computed for the subsequence $r_{i+1}, \ldots, r_j$ (ending at node s').

We process the previous steps in increasing order of l until l spans all the requests. Thus, the optimal cost is given by $\min_{s_{\text{final}} \in \{L,R\}} C[1, n, (s_{\text{init}}, s_{\text{final}})]$, where s_{init} is the initial server location.

Theorem 1. PAYC *(see Algorithm 1) computes the optimal contracts for Pay-as-You-Come model. The time complexity of* PAYC *is* $O(n^2(n + kq))$*, where* n *is the*

number of requests, k is the number of contract durations and q is the number of different bandwidth contracts.

Proof. The correctness follows by induction over the number of request l and by the optimal substructure property. Due to space constraints, we only sketch the proof. The claim is trivially true for sequences of one request (Lines 1–5). Consider the time interval from t_i to t_j with l requests, where $1 \leq i \leq j \leq n$ and $2 \leq l(= j - i + 1) \leq n$. This interval is split into two subintervals (Case I), or a long contract is bought that covers the entire interval (Case II). In *Case I*, we split the cost at time t_u with the server located at s'' such that the total cost $C[i, u, (s, s'')] + C[u + 1, j, (s'', s')]$ is minimized, where $i \leq u \leq j$ and $s'' \in \{L, R\}$. Since the number of requests in the two subintervals, $u - i + 1$ and $j - u$, are shorter than l, by the induction hypothesis, $C[i, u, (s, s'')]$ and $C[u + 1, j, (s'', s')]$ already store the optimal costs for these two intervals respectively. In *Case II*, we buy a long contract to cover the whole interval. Given a certain server location s'' at the start of the time t_{i+1}, $AM_m[i + 1, j, (s'', s')]$ already stores the optimal access and migration strategy cost for bandwidth b_m for interval $[t_{i+1}, t_j]$. Therefore, an optimal migration strategy for interval $[t_i, t_j]$ using bandwidth b_m can be obtained by adding r_i to the optimal strategies selected for the interval $[t_i + 1, t_j]$ and optimizing over the choice on whether to migrate the server to serve r_i or not (resulting in the two possible choices for s'', the position of the server right after satisfying request r_i).

Now we consider the time complexity of the PAYC algorithm. Clearly, the first phase of the algorithm requires time $O(nq)$. The second phase consists of three nested loop and has a complexity of $O(n^2 \cdot (n + kq))$. □

2.4.2 *Pay-as-You-Go*

Optimal solutions can also be computed for the Pay-as-You-Go model, and the algorithm PAYG is similar to the algorithm PAYC. As discussed above, in the Pay-as-You-Come model we need to decide when to migrate, which contracts to buy, and how much bandwidth to use. In the Pay-as-You-Go model, we still need to make a decision on when to migrate and how much bandwidth should be reserved, but we do not have to explicitly decide on a time contract. However, unlike the Pay-as-you-Come model, in the Pay-as-you-Go model, a bandwith b_m has to be chosen and fixed for satisfying the entire sequence of requests $r_i, \ldots, r_j$. Also, the contract cost in this model is directly dependent on the number of migrations of the server, and hence we explicitly have to keep track of this number.

Algorithm PAYG is listed in Algorithm 2. PAYG uses a new matrix $(A_m)_{n\times n\times 4}$ to store the access cost under a certain bandwidth $b_m,\ 1 \leq m \leq q$, and another matrix $(N_m)_{n\times n\times 4}$ is used to store the migration number for bandwidth b_m. A matrix $(C_m)_{n\times n\times 4}$ stores the total cost for bandwith b_m. In the entries of the new matrices, the elements $A_m[i, j, (s, s')]$ and $N_m[i, j, (s, s')]$ store the access cost and the number of migrations, respectively, for the optimal solution between time t_i and t_j with an initial server location s and a final server location s', where $s,\ s' \in \{L, R\}$. The entry $C_m[i, j, (s, s')]$ stores the total optimal cost within this time period for bandwith b_m.

The basic idea behind PAYG is to compute the optimal solution for a scenario where all the requests require the same bandwidth, and then choose the smallest cost among all the bandwidth options. PAYG starts off by computing the optimal costs for satisfying one request (Lines 1-5). Given the request r_i and the starting and ending server locations $s,\ s'$, the access cost is given by $D[s', r_i]$ which is 0 if the final server location s' and the request location r_i coincide, and 1 otherwise.

Algorithm 2 Algorithm PAYG

Input: Requests $<r_1, r_2, ..., r_n>$ at respective times $<t_1, t_2, ..., t_n>$.

Output: Minimum Cost.

1: **for** $i = 1$ **to** n **do**

2: **for** all pairs $(s, s') \in \{L, R\}^2$ and $1 \leq m \leq q$ **do**

3: $A_m[i, i, (s, s')] \leftarrow D[s', r_i]$

4: $N_m[i, i, (s, s')] \leftarrow D[s, s']$

5: $C_m[i, i, (s, s')] \leftarrow A_m[i, i, (s, s')] + S \cdot N_m[i, i, (s, s')]/b_m + f(D[s, s'], b_m)$

6: **for** $l = 2$ **to** n **do**

7: **for** $i = 1$ **to** $n - l + 1$ **do**

8: $j \leftarrow i + l - 1$

9: **for** all pairs$(s, s') \in \{L, R\}^2$ and $1 \leq m \leq q$ **do**

10: $C_m[i, j, (s, s')] \leftarrow \min_{i \leq u < j; s'' \in \{L,R\}} \{A_m[i, u, (s, s'')] + A_m[u + 1, j, (s'', s')] + S \cdot (N_m[i, u, (s, s'')] + N_m[u + 1, j, (s'', s')])/b_m + f((N_m[i, u, (s, s'')] + N_m[u + 1, j, (s'', s')]), b_m)\}$

11: Let (u, s'') be the parameter and location of request r_u at t_u that minimized Line 10.

12: $A_m[i, j, (s, s')] \leftarrow A_m[i, u, (s, s'')] + A_m[u + 1, j, (s'', s')]$

13: $N_m[i, j, (s, s')] \leftarrow N_m[i, u, (s, s'')] + N_m[u + 1, j, (s'', s')]$

14: **return** $\min_{s_{\text{final}} \in \{L,R\},\ 1 \leq m \leq q} C_m[1, n, (s_{\text{init}}, s_{\text{final}})]$

Meanwhile $D[s, s']$ will indicate that the server migrates to the other location to serve the current request if $D[s, s']$ is 1. Otherwise, there is no migration, and the starting and ending server locations s, s' describe the same node. We store the optimal solution in the entry $C_m[i, i, (s, s')]$ for each bandwidth b_m, where $C_m[i, i, (s, s')] = D[s', r_i] + S \cdot D[s, s']/b_m + f(D[s, s'], b_m)$.

Now PAYG iterates over the number of requests l (Line 6). For each value of l, we compute all the possible cases, as in Lines 7-13. First, we select from $[1, n-l-1]$ the value i denoting the index of the first of these l requests. Obviously, the index of the last of the l requests (denoted by j) would be $i+l-1$, as in Line 8. Assume that the server is located at node s at the time when the ith request occurs, and located at node s' at the end of the jth request, where $s, s' \in \{L, R\}$. We look for a way to split the duration such that the total cost $C_m[i, j, (s, s')]$ is minimized, as shown in Line 10. We use u, m, and s'' to denote the index of the request occurring at the chosen split point, the chosen bandwidth, and the location of the server (Line 11). Therefore, the total cost consists of the summation of the access costs of two subintervals, the summation of the migration costs of two subintervals, and a long contract cost covering the whole period. Here, the access cost is computed as $A_m[i, u, (s, s'')] + A_m[u+1, j, (s'', s')]$, the migration cost is computed as $(N_m[i, u, (s, s'')] + N_m[u+1, j, (s'', s')])/b_m$ and the contract cost is computed as $f(N_m[i, u, (s, s'')] + N_m[u+1, j, (s'', s')], b_m)$, for a certain bandwidth b_m. We store the access cost in $A_m[i, j, (s, s')]$ (Line 12) and the number of migrations in $N_m[i, j, (s, s')]$ (Line 13) for the current duration.

For each bandwidth b_m, we store the optimal solution to serve all the requests in C_m matrix. Thus the optimal cost is hence obtained by computing $\min_{s_{\text{final}} \in (L,R), 1 \le m \le q} C_m[1, n, (s_{\text{init}}, s_{\text{final}})]$ (Line 14).

The following claim follows by simple induction over the number of requests.

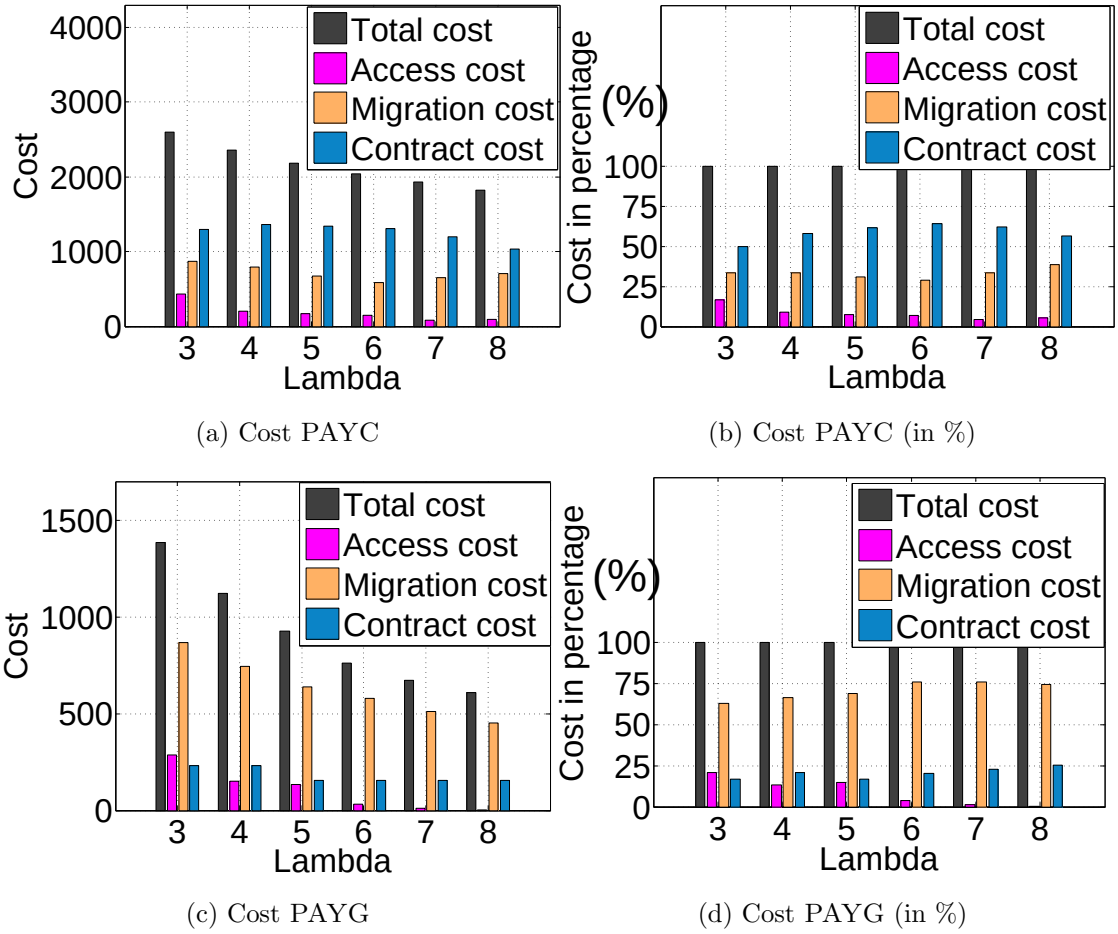
Theorem 2. PAYG *(see Algorithm 2) computes the optimal contracts for the Pay-as-You-Go model. The time complexity is $O(qn^3)$, where n is the number of requests and q is the number of different contract bandwidths.*

2.5 Simulation

The presented economical migration algorithms allow us to shed light on the properties of the two pricing models. We study three different discount functions $f_{\text{lin}}, f_{\text{sqrt}}, f_{\log}$ which offer cheaper contracts if longer (in terms of days) or larger (in terms of leased bandwidth) contracts are bought: f_{lin} is linear ("get twice as much for a 50% higher price"), f_{sqrt} grows according to a square root function and hence describes a steeper discount, and $f_{\log}$ even gives an even steeper logarithmic discount. For all three discount functions, the cost of a one-day contract with 50 Mbit/s bandwidth is the same, namely $f_i(1, 50) = 6$ for $i \in \{\text{lin, sqrt, log}\}$. Concretely, we use $f_{\text{lin}}(d_i, b_j) = 1.5 \cdot f_{\text{lin}}(d_i/2, b_j) = 1.5 \cdot f_{\text{lin}}(d_i, b_j/2) = 1.5^{(\lfloor \log d_i \rfloor + b_j/50 - 1)} \cdot f_{\text{lin}}(1, 50)$, $f_{\text{sqrt}}(d_i, b_j) = \sqrt{d_i b_j/50} \cdot f_{\text{sqrt}}(1, 50)$, and $f_{\log}(d_i, b_j) = \log(d_i b_j/50) \cdot f_{\log}(1, 50)$. We assume a server of size $S = 250$ MB, and we assume that the access cost for one remote request is five units (a request originating at the node where the service is located is free). We study a scenario where the provider offers two different bandwidth capacities, namely 50 Mbit/s and 100 Mbit/s, and four types of contract durations, namely 1, 30, 60 and 100 days (i.e., $\mathfrak{B} = \{50, 100\}$ and $\mathfrak{D} = \{1, 30, 60, 100\}$).

We study a simple request pattern where requests originate from two server locations L and R in turn, e.g., requests originating in Asia alternate with requests originating in the U.S..

Simplified Demand Scenario: We assume that requests alternate infinitely between the two sites L and R in the following manner: requests originate from one site (one per round) for a time interval duration which is chosen according to an exponential distribution with parameter λ, before requests originate from the opposite side again (according to the same distribution).



(a) Cost PAYC

(b) Cost PAYC (in %)

(c) Cost PAYG

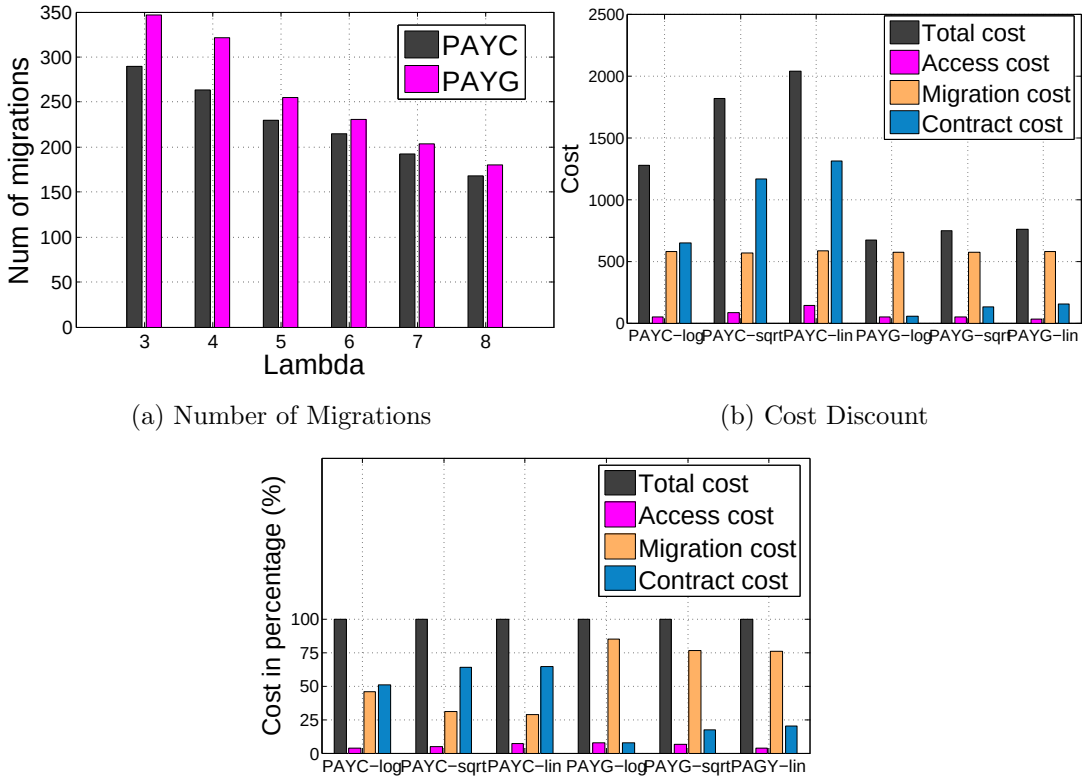
(d) Cost PAYG (in %)

Figure 2.1: Cost Distribution for PAYC and PAYG.

We simulate $n = 1500$ requests, and present the average over five runs for each experiment.

We discuss the following simulations in more detail.

Cost Distribution and Number of Migrations. We analyze how the cost distributes among the access cost, the migration cost, and the contract cost for the two algorithms PAYC and PAYG. All experiments discussed here are conducted under the natural f_{lin} discount function. Figure 2.1a shows the absolute costs of PAYC as a function of λ. We observe that the total cost and the access cost decrease for larger λ while the migration and contract stay much more stable. This is clear as requests



(a) Number of Migrations

(b) Cost Discount

(c) Cost Discount (in %)

Figure 2.2: Number of Migrations and Effect of Discount Function.

originating from one site for longer time periods render it worthwhile to migrate and buy longer contracts. The contract increases firstly and then decrease after some point, since the total migration numbers decrease and hence the contract cost is reduced. As the number of migrations decrease, the average number of migrations within a contract is also decreased. Therefore, PAYC will buy smaller bandwidth for such contract, which will result in larger migration costs(also shown in Table 2.2). Figure 2.1b presents the relative shares of the three costs. While the access costs approach zero for larger λ since the server is often at the right location, the contract costs and the migration costs stay stable since PAYC migrates a lot even for larger λ. The same results for PAYG are shown in Figures 2.1c and 2.1d, respectively. As a

first takeaway, we see that the cost distribution of Figure 2.1c defers from Figure 2.1a in that the total costs are lower, i.e., Pay-as-You-Go is always the cheaper option than Pay-as-You-Come pricing for the customer. Also note that in contrast to the Pay-as-You-Come model, the migrations constitute a larger share of the overall costs, since the contract cost is given by the number of migrations and the amount of leased bandwidth under the discount function; hence the contract cost is lower than the one of PAYC for the same number of migrations. Moreover, there are relatively more frequent migrations under the PAYG model, see Figure 2.2a, which also explains the lower access costs (i.e., this improves QoS experienced by the users). Regarding the relative cost shares (Figure 2.1d), we can see that the percentage for the access cost is decreasing while the percentages for the migration cost and the contract cost are increasing slowly. Again, when λ is large enough and the requests become more local, since migrations only occur at the beginning of each interval, the number of migrations (as well as all three cost components) eventually decreases.

Contract Distribution. Different pricing models and scenarios result in different types and combinations of contracts, and it is interesting to study the frequency (or popularity) distribution of the contracts. Table 2.1 reports on the average number of the contracts as a function of λ, for different contract durations and bandwidths, under the PAYC algorithm and for discount function f_{lin}. We see that when λ is small and migrations are dense, longer duration contracts occur frequently since the server migrates often. However, as λ increases, all lengths of contracts decrease. As λ increases, the average number of migrations in a contract decreases and hence the smaller bandwidth will benefit more than the larger one. Therefore, it turns out to buy more contracts with smaller bandwidth. This can also be seen in Table 2.2 which records the average number of migrations in different contracts accordingly (average over five runs).

Table 2.1: Distribution of Purchased Contracts (Discount Function f_{lin}).

Dur-Bw \ λ	3	4	5	6	7	8
1-50	11.2	8	15.4	13.8	18.4	39.2
60-50	0	0	0	0	2.4	0.8
60-100	1.4	2	1.4	2.8	1	0.4
100-50	0	0	0	0.6	2	5.4
100-100	11	11	11.2	10	7.6	3.4

Table 2.2: Number of Migrations for Each Contract (Discount Function f_{lin}).

Dur-Bw \ λ	3	4	5	6	7	8
1-50	1	1	1	1	1	1
60-50	0	0	0	0	8,5	0
60-100	17.67	14	13.5	11.5	0	0
100-50	0	0	0	13	13	12.57
100-100	27.33	23.58	19.45	17.33	15	14.5

Impact of Discount Function. Finally, let us compare the different discount functions in more detail. Figure 2.2b and Figure 2.2c explore the absolute and relative (in %) cost distributions for PAYC and PAYG under different discount functions. Clearly, the higher the discount, the smaller the total cost. Moreover, not surprisingly the performance of PAYG is always better than that of PAYC since the total cost is

less for PAYG compared to that for PAYC. However, the difference of the costs for the two models is smaller for higher discounts, i.e., the difference for the logarithmic discount function is smaller than for a discount function which follows a square root.

2.6 A First Look at Online Migration

Although the main focus of this work is on predictable demand scenarios and offline algorithms, in this section, we want to initiate the discussion of online algorithms. The online discussion builds upon our offline results in two respects: First, some algorithmic techniques from the offline variant may be used also for the online variants. For example, an online algorithm may try to predict the future from the past, and apply an optimal offline algorithm on a sequence of recent past requests in order to make decisions on how to deal with upcoming requests. Second, offline algorithms are often needed to evaluate the performance of an online algorithm. The ratio of the cost of an online algorithm divided by the cost of an optimal offline algorithm is also known as the *competitive ratio* [5].

Both online algorithms presented in the following are inspired by the (optimal) offline variants and seek to amortize costs over time. To simplify the presentation, we assume a constant bandwidth scenario.

OnC: The online Pay-as-You-Come algorithm OnC tracks the access costs it incurs at the current location using a counter C. Once the counter exceeds the migration cost (given by the server size divided by the bandwidth), OnC migrates the server and resets C. If there is currently no contract available for migration, OnC checks whether a contract longer than the most recently used contract would have been better *for the past requests*. Concretely, OnC checks longer contracts one by one (in increasing order of length) and compares their costs in the corresponding

intervals (starting from the last migration) to the cost OnC incurred during that time period. As soon as a better contract is found, it is chosen. Otherwise, OnC checks whether a contract shorter than the most recent contract should be chosen. The following heuristic is applied: OnC checks whether during the last contract, the number of migrations was larger in the first half or the second half of the contract time interval. In case of the first half, OnC will buy the shorter contract; otherwise, OnC chooses the same contract as last time.

Now let us discuss a simple online algorithm OnG for the Pay-as-You-Go model. Since the customers only need to pay for the resources actually consumed, OnG just needs to decide when to migrate.

OnG: Let the counter C_1 record the number of the migrations performed so far and let the counter C_2 denote the total access costs. If the access cost C_2 reaches the migration cost plus marginal migration contract costs (i.e., $f(C_1+1,b)-f(C_1,b)$, for bandwidth b), OnG migrates the server, increments counter C_1, and resets counter C_2.

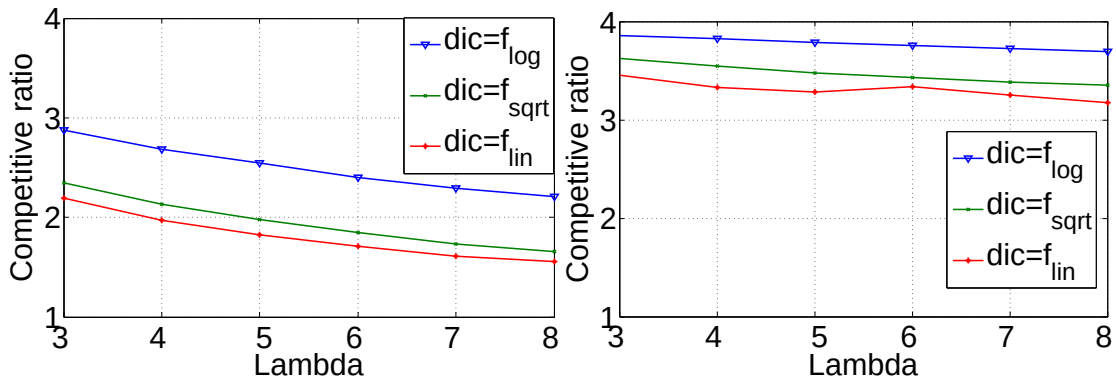
Given our optimal offline algorithms, it is interesting to study the *competitive ratio* of OnC and OnG. We study three different discount functions $f_{\text{lin}}, f_{\text{sqrt}}, f_{\log}$ which offer cheaper contracts if longer (in terms of days) or larger (in terms of leased bandwidth) contracts are bought: f_{lin} is linear ("get twice as much for a 50% higher price"), f_{sqrt} grows according to a square root function and hence describes a steeper discount, and $f_{\log}$ even gives an even steeper logarithmic discount. For all three discount functions, the cost of a one-day contract with 50 Mbit/s bandwidth is the same, namely $f_i(1,50)=6$ for $i \in \{\text{lin}, \text{sqrt}, \log\}$. Concretely, we use $f_{\text{lin}}(d_i,b_j)=1.5\cdot$

$f_{\text{lin}}(d_i/2, b_j) = 1.5 \cdot f_{\text{lin}}(d_i, b_j/2) = 1.5^{(\lfloor \log d_i \rfloor + b_j/50 - 1)} \cdot f_{\text{lin}}(1, 50)$, $f_{\text{sqrt}}(d_i, b_j) = \sqrt{d_i b_j/50} \cdot f_{\text{sqrt}}(1, 50)$, and $f_{\log}(d_i, b_j) = \log(d_i b_j/50) \cdot f_{\log}(1, 50)$. We assume a server of size $S = 250$ MB, and we assume that the access cost for one remote request is five units (a request originating at the node where the service is located is free). We study a scenario where the provider offers two different bandwidth capacities, namely 50 Mbit/s and 100 Mbit/s, and four types of contract durations, namely 1, 30, 60 and 100 days (i.e., $\mathfrak{B} = \{50, 100\}$ and $\mathfrak{D} = \{1, 30, 60, 100\}$).

The competitive ratios for OnC and OnG are presented in Figure 2.3. We observe that the ratios for both algorithms are relatively small (between 1.5 and 4) and decrease for larger λ (lower dynamics). This can be explained by the fact that with higher λ, requests remain more local and migration patterns more obvious. A second takeaway is that the competitive ratio for the lowest discount function f_{lin} is best, while higher discounts like f_{log} are handled worse by our online algorithms. Especially in the Pay-as-You-Come model, our online algorithm has more difficulties to deal with high discounts, as it tends to buy too many short contracts (OnC migrates more often than the offline algorithm). Also under Pay-as-You-Go pricing, the offline algorithm can exploit discounts relatively better, although to a lesser extent. (The offline algorithm migrates relatively more frequently for higher discounts.)

2.7 Conclusion

There is a large body of literature on economical aspects of cloud computing, but much less is known about efficient (virtual) network pricing. Interestingly, while cloud (or node) resources are often priced according to a flexible *per-use* or pay-as-you-go policy, networking services such as MPLS connectivity are often charged according to usage-independent, time-based policies [40]. This is particularly surprising as network demand is likely to exhibit a higher variance over time than, e.g., storage resources.



(a) Competitive Ratio for OnC.

(b) Competitive Ratio for OnG.

Figure 2.3: Effect of Discount Function on Competitive Ratio. We Simulate 1500 Requests and Present the Average over Five Runs.

For instance, distributed SAP systems may be fully synchronized only sporadically (but then lead to high network loads), whereas the resource requirements of, e.g., a mail service normally grows monotonically over time.

We understand this work as a first step to study the effect of virtual network pricing policies on *service migration*. We focused on the offline setting where demand patterns are given (e.g., describe regular time-of-day or commuter effects). Such online algorithms can also be useful to evaluate the competitive ratio of online algorithms in simulations. We presented two optimal algorithms for efficient service migration in different economic settings. We believe that the used algorithmic techniques are relatively general and can be extended to more complex scenarios, e.g., to networks supporting live migration or more complex virtual network topologies.

Chapter 3

COMPETITIVE STRATEGIES FOR ONLINE CLOUD RESOURCE ALLOCATION WITH DISCOUNTS: THE 2-DIMENSIONAL PARKING PERMIT PROBLEM

3.1 Introducation

As the Internet becomes increasingly virtualized, resources can be allocated more flexibly and at large scale. Virtualization not only introduces an Internet-wide resource market, but also new business models. For example, a startup company running a webservice no longer needs to invest in its own infrastructure, but can dynamically lease cloud resources to provide the service to its users in a cost-efficient manner. Also a hybrid model is possible where the cloud resources are just used to complement a limited own infrastructure in peak demand times (a.k.a. "cloud bursting"). New business models are also introduced by resource brokering opportunities: a broker may lease a large amount of resources from different providers and resell them to its customers (at a higher price).

This work studies the problem of a (cloud) *customer* who rents resource bundles from a (cloud) *provider*, in order to offer a certain service to its users (or to resell the resources). The customer is faced with the challenge that its resource demand (e.g., the popularity of its webservice) is not known in advance. In order to ensure that its resource demand is satisfied at any time, and in order to avoid a costly overprovisioning of the service, additional resources must be bought in an *online* manner. The online resource allocation problem may further be complicated by the fact that the provider offers *discounts* for larger and longer resource contracts.

The goal of the customer is to come up with a smart resource renting strategy to satisfy its dynamic and unpredictable resource demand, while minimizing the overall costs of the bought resource bundles.

Our Contributions. This work shows that at the heart of efficient cloud resource allocation lies a fundamental algorithmic problem, and makes the following contributions. We first observe that the problem of renting a single resource over time can be seen as a *2-dimensional* variant of the well-known *online Parking Permit Problem* (PPP). While in the classic parking permit problem, only the *contract durations* need to be chosen, in the 2-dimensional variant PPP^2 introduced in this work, also the *resource rates* are subject to optimization.

Our main contribution is the deterministic online algorithm On2D whose performance is close to the one of a clairvoyant optimal offline algorithm which knows the entire resource demand in advance: On2D provably achieves a competitive ratio of $O(k)$, where k is the total number of available resource contracts; this is asymptotically optimal in the sense that there cannot exist any deterministic online algorithm with competitive ratio $o(k)$.

We also give a constructive proof that the offline variant of the PPP^2 problem can be solved in polynomial time, by presenting a dynamic programming algorithm Off2D accordingly. To the best of our knowledge, Off2D is also the first offline algorithm to efficiently solve PPP and PPP^2 for long enough request sequences σ. Off2D is used as a subroutine in On2D.

Finally, we show that our algorithms and results also generalize to multi-resource scenarios, i.e., to higher-dimensional parking permit problems.

Work Organization. The remainder of this chapter is organized as follows. Section 3.2 reviews related work. Section 3.3 formally introduces our model. Section 3.4 presents our online algorithm. In detail, we discuss an example and provides intuition

for the analysis, and present the general analysis in this section . We show that our algorithm is almost optimal by deriving a lower bound in Section 3.5. Section 3.6 presents a polynomial-time optimal offline algorithm, and Section 3.7 shows how to extend our results from 2-dimensions to D-dimensions (for a constant D). Finally, we conclude our work in Section 3.9.

3.2 Related Work

Cost reductions (due to economy-of-scale effects) are arguably the main motivation behind today's trend to out-source infrastructure and software to the cloud. Accordingly, a large body of literature in the field focuses on resource allocation and scheduling problems: the more virtual services that can be multiplexed on a given physical network, the better the resource sharing and hence provider revenue. For a good overview of the field, we refer the reader to the surveys [4, 14]. We will structure the discussion of related works according to the following categories: cloud network performance, economic models, and online optimization algorithms.

Cloud Network Performance. This work assumed that the service of the cloud customer comes with hard resource requirements, which is motivated by the varying and often unpredictable performance in today's oversubscribed datacenters. Indeed, several studies have shown that services and applications without strict performance isolation suffer from poor performance. [32] Accordingly, researchers have proposed to make resource reservations more explicit and also provide bandwidth isolation. For example, SecondNet [22] and Oktopus [6] introduce the notion of virtual networks that provide the illusion of a dedicated network, specified as a graph resp. Hose model. Proteus [46] generalizes these concepts in the time-dimension, supporting a flexible interleaving of reservations over time. In order to deal with unexpected external interference and failures, Q-Clouds [33] saves certain resource which can

be allocated to mask interrupting events. For a good overview on cloud network performance issues, we refer the reader to [32]. Virtual services and networks have also been investigated outside datacenters. Especially Software-Defined Networking and Network Function Virtualization offer appealing innovation opportunities to ISPs, and allow the introduction of new services in the network core. [13, 48]

Economic Roles and Challenges. Compared to the algorithmic problems of the resource allocation and scheduling, the economical aspects are less well-understood. Different economical cloud models have been proposed and compared by various authors, e.g., by Armbrust et al. [30], Pal et al. [34], or Dash et al. [17]. Some of the studied pricing models have their origins in the context of ISP-customer relationships [42] and are also related to classic economic problems [20]. An interesting tradeoff between time and price has been studied in [25], from a scheduling complexity perspective. More generally, there are several interesting proposals for novel adaptive resource and spot market pricing schemes, e.g., [1].

Our role model is motivated by the network virtualization architecture presented by Schaffrath et al. [39]: the authors identify stakeholders and new economical roles, and distinguish between a physical infrastructure provider, a virtual network provider (i.e., resource reseller or broker), a virtual network operator and a service provider. In the terminology used in this work, the physical infrastructure provider corresponds to our *provider*, and the virtual network provider corresponds to the *customer*. The customer and broker role more arises in virtualization architectures allowing for subrenting and recursion [12, 39].

Online Algorithms and the Parking Permit Problem. The work closest to ours is the Parking Permit Problem (PPP) work [31]. Analogously to the classic ski-rental problem, PPP is the archetype for online problems where purchases have time durations which expire regardless of whether the purchase is used or not. Formally,

PPP specifies a set of k different types of parking permits of a certain price π_i and duration d_i. In [31], Meyerson presents an asymptotically optimal deterministic online algorithm with competitive ratio $O(k)$ (together with a lower bound of $\Omega(k)$). The work also discusses randomized algorithms and an application to Steiner forests. While we can build upon some of the techniques in [31], the rate dimension renders the problem rather different in nature, both from an online and an offline algorithm perspective.

To the best of our knowledge, there does not exist any work on online algorithms for two-dimensional, time and rate based resource rental problems.

More generally, online problems with discounts have also been studied in various contexts already, especially rental [47] and ticket purchase problems [19]. There also exist online algorithms with provable competitive ratios for other optimization problems arising in the context of cloud computing and network virtualization, such as online access control [7, 18] or service migration [9].

3.3 Model

We attend to the following setting (for an illustration, see Figure 3.1). We consider a customer with a dynamically changing resource demand. We model this resource demand as a sequence $\sigma = (\sigma_t)_t$, where σ_t refers to the resource demand at time t. We use $\widehat{\sigma}$ to denote $\max_t \sigma_t$. The customer is faced with the challenge that its future resource requirements are hard to predict, and may change in a worst-case manner over time: We are in the realm of *online algorithms and competitive analysis*.

In order to cover its resource demand, the customer buys resource contracts from a (cloud) provider. For ease of presentation, we will focus on a single resource scenario for most of our work; however, we will also show that our results can be extended to multi-resource scenarios. Concretely, we assume that the provider offers different

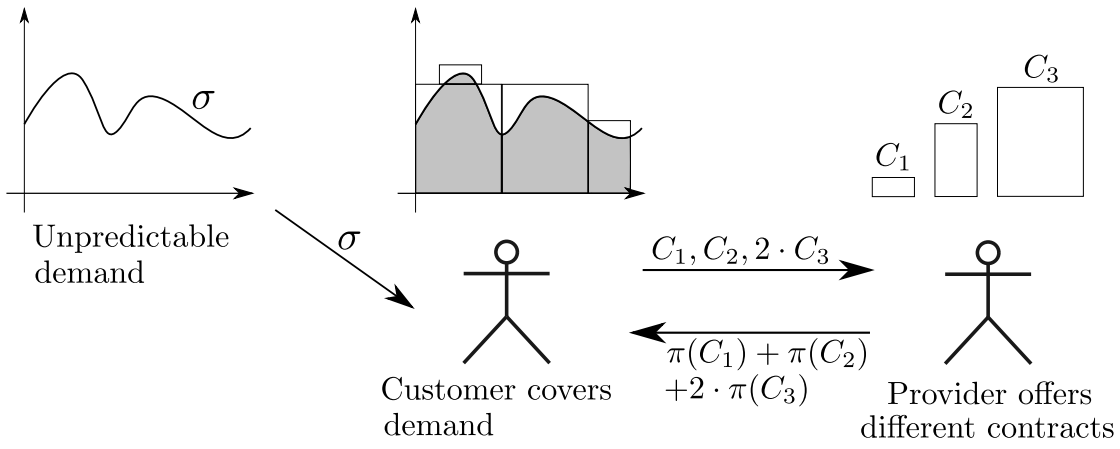


Figure 3.1: Overview of the Model: A Customer Has to Cover Its Resource Demand σ by Buying Contracts $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ from the Provider. Larger Contracts C_i Come with a Price Discount (Price $\pi(C_i)$).

resource contracts $C(r, d)$ of *resource rate* r and *duration* d. We will refer to the set of available contracts by $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. Each contract type has a *price* $\pi(C) = \pi(r, d)$, which depends on its rate $r(C) = r$ and its duration $d(C) = d$. We will assume that resource contracts have a monotonically increasing rate-duration product $r \times d$, and will denote by C_i the i^{th} largest contract.

A specific contract instance of type C_i will be denoted by $C_i^{(j)}$, for some index j. Each instance $C_i^{(j)}$ of contract type $C_i(r, d)$ has a specific start time $t_i^{(j)}$, and we will sometimes refer to a contract instance by $C_i^{(j)}(t_i^{(j)}, r_i, d_i)$. The identifiers are needed since multiple contracts of the same type can be stacked in our model, but will be omitted if the contract is clear from the context.

We will make three simplifying assumptions:

A1 *Monotonic Prices:* Prices are monotonically increasing, i.e., larger contracts are more expensive than shorter contracts: $\pi(C_{i-1}) < \pi(C_i)$ since $r_{i-1} \times d_{i-1} < r_i \times d_i$ for any i.

A2 *Multiplicity:* The duration and resource rate of a contract of type $C_i(r_i, d_i)$ are perfect multiples of the duration and rate, respectively, of contract $C_{i-1}(r_{i-1}, d_{i-1})$. That is, we assume that $d_i = x \cdot d_{i-1}$ and $r_i = y \cdot r_{i-1}$ for fixed bases $x, y \geq 2$. Moreover, *w.l.o.g.*[1] we will assume that the smallest contract has $d_1 = 1$ and $r_1 = 1$.

A3 *Intervals:* We assume that a contract of duration d can only be bought at time $t_0 + i \cdot d$, where $t_0 = 0$ is the start time.

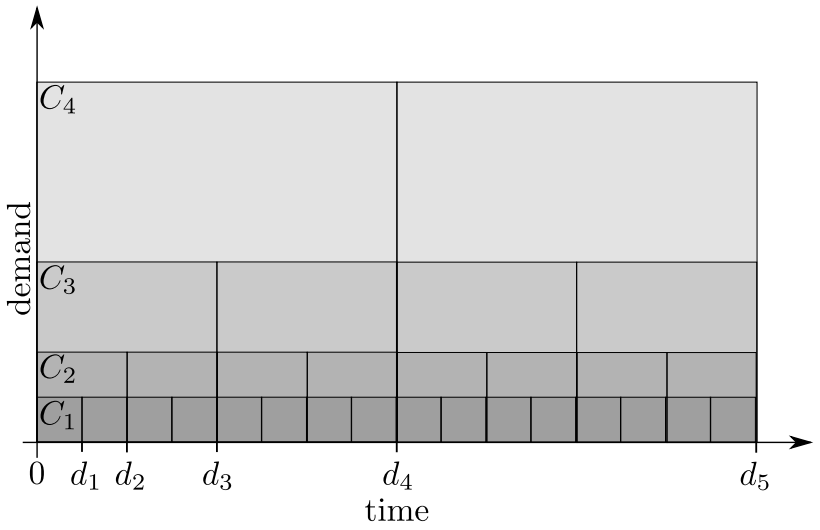


Figure 3.2: The Interval Model with Four Different Contracts: Each Contract $C(r, d)$ Can only Be Bought at Time $t_0 + i \cdot d$.

Assumption A1 is natural: contracts which are dominated by larger, cheaper contracts may simply be ignored. Assumption A2 restricts the variety of available contracts the customer can choose from, and constitutes the main simplification made in this work. Assumption A3 mainly serves the ease of presentation: as we will prove in this work (and as it has already been shown for the classic Parking Permit

[1]Without loss of generality.

Problem [31]), an offline or online algorithm limited by the interval model is at most a factor of two off the respective optimal solution in the general model.

Now, let On be some online algorithm, let $\mathcal{C}_t(\text{On})$ denote the set of contracts bought according to On at time t and let $\mathcal{C}_{\leq t}(\text{On})$ denote the set of contracts bought according to On through time t. We will use the notation $\mathcal{C}^*_t(\text{On}) \subseteq \mathcal{C}_{\leq t}(\text{On})$ to describe the set of *active* contracts at time t: i.e., contracts $C_i(t_i, r_i, d_i)$ bought by On with $t_i \leq t < t_i + d_i$. Likewise we denote the set of contracts bought by an optimal offline algorithm Off to cover the demand prefix $\sigma_1, ..., \sigma_t$ until time t by $\mathcal{C}_{\leq t}(\text{Off})$.

Since a correct algorithm must ensure that there are always sufficient resources to cover the current demand, the invariant $\sum_{C(r,d)\in\mathcal{C}^*_t(\text{On})} r \geq \sigma_t$ must hold at any moment of time t. We will use the *one-lookahead model* [11] frequently considered in online literature, and allow an online algorithm to buy a contract at time t *before* serving the request σ_t; however, On does not have any information at all about $\sigma_{t'}$ for $t' > t$.

The goal is to minimize the overall price $\pi_\sigma(\text{On}) = \sum_{C\in\mathcal{C}_{\leq t}(\text{On})} \pi(C)$. More specifically, we seek to be competitive against an optimal offline algorithm and want to minimize the *competitive ratio* ρ of On: We compare the price $\pi_\sigma(\text{On})$ of the online algorithm On under the external (online) demand σ, to the price $\pi_\sigma(\text{Off})$ paid by an optimal offline algorithm Off, which knows the entire demand σ in advance. Formally, we assume a worst-case perspective and want to minimize the (strict) competitive ratio ρ for any σ: $\rho = \max_\sigma \pi_\sigma(\text{On})/\pi_\sigma(\text{Off})$.

We are interested in long demand sequences σ; in particular, we will assume that the length of σ, $|\sigma|$, is at least as large as the largest single demand σ_t.

Our problem is an interesting variant of the classic Parking Permit Problem PPP [31], which we review quickly in the following. In PPP, a driver has to choose

between k parking permits of different durations and costs in order to satisfy all of his/her parking needs while minimizing the total cost paid. More precisely, the driver has a sequence of days when he/she needs a parking space at a parking garage and there are k different parking permits, where each parking permit P_i allows the driver to use one parking space for d_i consecutive days at a cost of c_i. In the online version of the problem, the sequence of days when the driver will need a parking space is not known in advance.

3.4 Competitive Online Algorithm

This section presents the deterministic online algorithm On2D for the PPP^2 problem. As a subroutine, in order to determine which contracts to buy at time t, On2D uses an optimal offline algorithm Off2D that computes optimal contracts for a prefix $\sigma_{\leq t}$ of the demand through time t. In this section, we will treat Off2D as a black box, but will describe a polynomial-time construction later in Section 3.6 of this chapter. Moreover, for ease of presentation, by default we will focus on the *Interval Model*. (The competitive ratio increases at most by a factor 2 in the general model.)

3.4.1 The Algorithm

In order to formally describe and analyze our algorithm, we propose a scheme that assigns bought contracts to the 2-dimensional *time-demand plane* (see Figure 3.2 for an illustration). Our model requires that each point below the demand curve σ is covered by a contract, i.e., the mapping of contracts to demand points must be surjective.

We pursue the following strategy to assign contracts to the time-demand plane: at any time t, we order the set of active contracts by their duration, and stack the active contracts in such a way that longer contracts are embedded lower in the plane,

i.e., the longest running contract $C_i(r_i, d_i)$ covers the demand from 1 to r_i, the next shorter contract $C_j(r_j, d_j)$ then covers the demand $r_i + 1$ to $r_i + r_j$, and so on. This guarantees a unique mapping of a demand point $p(time, demand)$ at time t to a contract C_i for the offline algorithm.

Algorithm 3 Online Algorithm On2D

Input: Demand prefix $\sigma_{\leq t} = \sigma_1, \sigma_2, ..., \sigma_t$; set of contracts $\mathcal{C}_{\leq t-1}(\text{On2D})$ bought by On2D through time $t-1$

Output: Contracts to be bought at time t: $\mathcal{C}_t(\text{On2D})$

1: $\mathcal{C}_t(\text{Off2D}) \leftarrow \text{Off2D}(\sigma_1, \sigma_2, ..., \sigma_t)$
2: **for** $C \in \mathcal{C}_t(\text{Off2D})$ **do**
3: **if** $\exists$ demand point p covered by C such that p is not covered by $\mathcal{C}_{\leq t-1}(\text{On2D})$ **then**
4: $\mathcal{C}_t(\text{On2D}).add(C)$
5: **return** $\mathcal{C}_t(\text{On2D})$

Our online algorithm On2D (see Algorithm 3) is based on an oracle Off2D computing optimal offline solutions for the demand *so far*. On2D uses these solutions to purchase contracts at time t. Concretely, On2D mimics the offline algorithm in an efficient way, in the sense that it only buys the optimal offline contracts covering time t if the corresponding demand is not already covered by contracts bought previously by On2D: At each time t, On2D compares the set of previously bought contracts $\mathcal{C}_{\leq t-1}(\text{On2D})$ with the set of contracts $\mathcal{C}_t(\text{Off2D})$ that Off2D would buy for an offline demand sequence $\sigma_1, ..., \sigma_t$; On2D then only buys the contracts $C \in \mathcal{C}_t(\text{Off2D})$ for the demand at time t that is not covered by $\mathcal{C}_{\leq t-1}(\text{On2D})$.

3.4.2 Case Study

In order to provide some intuition of the behavior of On2D, as a case study, we consider the special scenario where contracts are perfect squares, i.e., $C_i = (2^{i-1}, 2^{i-1})$, and where the contract prices have a specific discount structure, namely $\pi(C_i) = 2 \cdot C_{i-1}$, with $\pi(C_1) = 1$. This price function ensures that Off2D will buy at most one C_j contract before it is worthwhile to buy the next larger contract C_{j+1} for the given time interval.

Let us now study the maximal cumulative price $\Pi(C_i)$. It is easy to see that under the price function above, the demand sequence σ with a constant demand of one unit per time, maximizes $\Pi(C_i)$ for $C_i = (2^{i-1}, 2^{i-1})$ and $\pi(C_i) = 2 \cdot C_{i-1}$: higher demands imply missed opportunities to charge On2D for smaller contracts, as already a demand of two at given time t renders it worthwhile to buy C_2, and a demand of four renders it worthwhile to buy C_3, etc.

With the given demand σ, Off2D will end up buying each of the smaller contracts once before it buys the next larger contract. The cumulative price derived from σ according to this behavior is $\Pi(C_i) = \sum_{j=1}^{i-1} \Pi(C_j) + \pi(C_i)$. We prove this claim by induction over the contract types i. For the base case $i = 1$, $\Pi(C_i) = \pi(C_i)$ holds trivially. Assuming the induction hypothesis for i we have:

$$\begin{aligned}
\Pi(C_{i+1}) &= \sum_{j=1}^{i} \Pi(C_j) + \pi(C_{i+1}) \\
&= \sum_{j=1}^{i-1} \Pi(C_j) + \Pi(C_i) + \pi(C_{i+1}) \\
&\overset{IH}{=} \sum_{j=1}^{i-1} \Pi(C_j) + \sum_{j=1}^{i-1} \Pi(C_j) + \pi(C_i) + \pi(C_{i+1})
\end{aligned}$$

Due to the induction hypothesis, the cost of a quarter of 2^{i+1}x2^{i+1} is maximized for $\sum_{j=1}^{i-1} \Pi(C_j) + \pi(C_i)$. In order to maximize the cost in the second quarter (at the

bottom of the time-demand plane) OFF2D would need to buy $\sum_{j=1}^{i-1} \Pi(C_j)$ again and instead of buying a second contract C_i the pricing scheme would lead to the purchase of contract C_{i+1}. Therefore buying the same contracts again (despite C_i) must lead to $\Pi(C_{i+1}) = \sum_{j=1}^{i} \Pi(C_j) + \pi(C_{i+1})$.

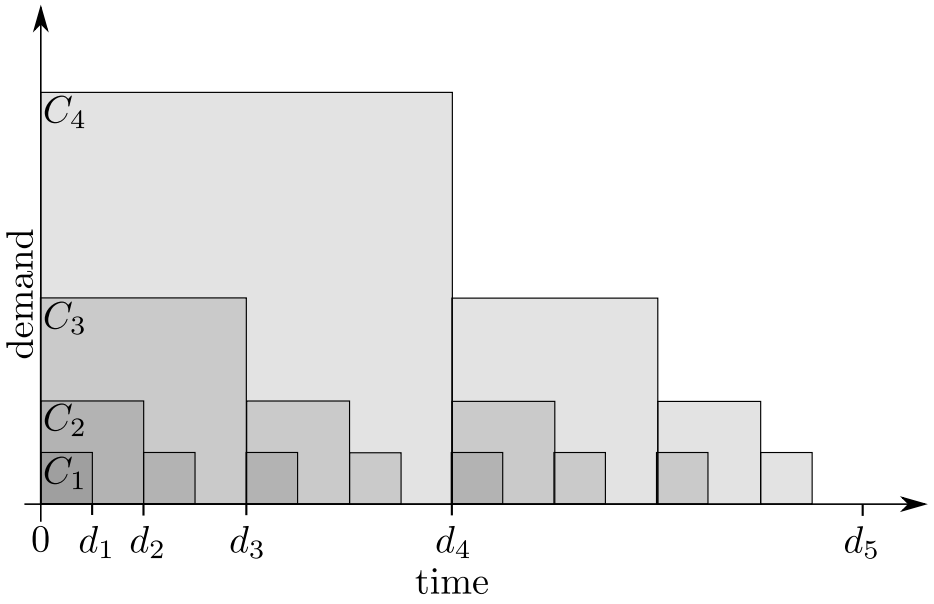


Figure 3.3: Worst-case Example Where $\sigma_t = 1 \ \ \forall t$. While OFF2D at Time Point d_5 Buys a Single Contract C_5, ON2D Is Forced to Buy all the Depicted Contracts, in Addition to C_5. For Instance, Buys C_1 in Every Second Time Step.

In conclusion, for the considered price function, we have derived a worst-case sequence σ, that shows that ON2D achieves a k-competitive ratio.

Theorem 3. *For the special setting considered in our case study,* ON2D *is k-competitive.*

Proof. Consider the discussed worst-case sequence σ, where ON2D has to buy every contract (total cost $\Pi(C_i)$) while OFF2D can simply buy C_i at price $\pi(C_i)$. We can show that $\Pi(C_i) \leq i \cdot \pi(C_i)$ and hence $\Pi(C_i) \leq k \cdot \pi(C_i)$. According to the observed behavior of OFF2D, every second contract bought by ON2D is C_1 (2^{i-2} times), every

fourth is C_2 (2^{i-3} times), etc., and finally ON2D also buys C_i. See Figure 3.3 for an example. Thus,

$$\begin{aligned}\Pi(C_i) &= 2^{i-2} \cdot \pi(C_1) + 2^{i-3} \cdot \pi(C_2) + \cdots + 1 \cdot \pi(C_i) \\ &= 2^{i-2} \cdot 2^0 + 2^{i-3} \cdot 2^1 + \cdots + 2^{i-i} \cdot 2^{i-2} + 1 \cdot 2^{i-1} \\ &\leq 2^{i-1} + 2^{i-1} + 2^{i-1} + \cdots + 2^{i-1} + 2^{i-1} \\ &= i \cdot 2^{i-1} = i \cdot \pi(C_i)\end{aligned}$$

□

3.4.3 General Analysis

With these intuitions in mind, we now present a general analysis of ON2D. First, we derive some simple properties of the contracts bought by the optimal offline algorithm OFF2D over time. Let us fix an arbitrary *demand point* p, i.e., a point below the σ-curve in the time-demand plane. We make the following claim: if p is covered by a certain contract C in $\mathcal{C}_{\leq t}(\text{OFF2D})$, p will never be covered by a smaller contract C' in $\mathcal{C}_{t'}(\text{OFF2D})$ for any $t' > t$. In other words, when considering a longer offline demand sequence $\sigma_1, \ldots, \sigma_{t'}$, OFF2D will never buy a smaller contract than C to cover the demand point p. This property of "growing contracts" together with the assumption of disjoint intervals motivates the notion of *contract independence*, which we formalize in the lemma below:

Lemma 1 (Contract Independence)**.** *Consider a demand point p_i covered by contract $C_i \in \mathcal{C}_{\leq t}(\text{OFF2D})$ and a demand point p_j covered by a distinct contract $C_j \in \mathcal{C}_{\leq t}(\text{OFF2D})$. Then there does not exist a contract $C \in \mathcal{C}_{t'}(\text{OFF2D})$ for any $t' < t$ such that p_i, p_j are covered by C. We say that the two contracts C_i and C_j are* independent.

Independence between contracts is always trivially ensured in our model. This enables a simple characterization of the scenarios maximizing the competitive ratio.

Lemma 2. *The competitive ratio is maximized in a scenario where* Off2D *buys only one contract to satisfy the entire demand σ.*

Proof. By contradiction. Assume Off2D buys more than one contract, say C_i and C_j. Now assume that over time, On2D buys a set of (possibly smaller) contracts $C_{i'}, C_{i''}, \ldots$ to cover the demand points of C_i and $C_{j'}, C_{j''}, \ldots$ to cover the demand points of C_j. Thus, On2D pays $\pi(C_i) + \pi(C_{i'}) + \ldots$ and $\pi(C_j) + \pi(C_{j'}) + \ldots$ whereas Off2D pays $\pi(C_i)$ and $\pi(C_j)$; the resulting competitive ratio is $\rho_{C_i} = (\pi(C_i) + \pi(C_{i'}) + \ldots)/\pi(C_i)$ for the C_i part and $\rho_{C_j} = (\pi(C_j) + \pi(C_{j'}) + \ldots)/\pi(C_j)$ respectively. Since all contracts in Off2D are independent, the competitive ratio ρ of Off2D will be $\max\{\rho_{C_i}, \rho_{C_j}\}$ which would also be achieved if the larger contract was the only one bought by Off2D. □

We hence want to show that On2D will never buy too many small contracts to cover a demand for which Off2D would later only buy one contract. Concretely, let us fix any contract $C_i \in \mathcal{C}_{\leq t}(\text{Off2D})$, and let us study the set of contracts S bought by On2D during the time interval $[0, t)$ which overlap with C_i in the time-demand plane. Recall that S will only contain distinct instances of the contracts (since On2D does not buy "repeated" contracts) and it will be contained in $\cup_{t' < t} \mathcal{C}_{t'}(\text{Off2D})$. By the interval and independence property, we know that contracts in S are all "inside" C_i, i.e., do not exceed its boundary in the plane. Accordingly, we can compute an upper bound on the maximum cumulative price spent on contracts in S by On2D while Off2D at time t only bought a single contract C_i at price π. In the following, let us refer to this cumulative price paid by On2D by $\Pi(C_i) = \sum_{C \in S} \pi(C)$.

Lemma 3. *The maximum cumulative price paid by* On2D *to cover a contract* C_i, $\Pi(C_i)$, *is less than or equal to* $i \cdot \pi(C_i)$, *for any* $i \geq 0$.

Proof. Consider a contract $C_i \in \mathcal{C}_{\leq t}(\text{Off2D})$ and S as defined above. Let ℓ be such that On2D has bought ℓ contracts C_{i-1} to cover the area of C_i during time $[0, t)$, where $0 \leq \ell \leq \frac{\pi(C_i)}{\pi(C_{i-1})}$. For all other $C \in S$, we must have $C \in \{C_1, \ldots, C_{i-2}\}$. Let $S' = \{C \in S,\ s.t.\ C \in \{C_1, \ldots, C_{i-2}\}\}$. Hence we have $\sum_{C \in S'} \pi(C) \leq \pi(C_i) - \ell \cdot \pi(C_{i-1})$, since the area covered by all contracts in S is at most equal to the area covered by C_i, and given Assumption A1. We argue by induction on i.

Base case $i = 1$*:* If there is just one type of contract C_1, the online algorithm will buy the same contracts as the offline algorithm, and the claim holds trivially.

Inductive step $i > 1$*:* Assuming the induction hypothesis holds for all $j < i$, we have:

$$
\begin{aligned}
\Pi(C_i) &= \ell \cdot \Pi(C_{i-1}) + \pi(C_i) + \sum_{C_j^{(p)} \in S'} \Pi(C_j^{(p)}) \\
&\leq \ell \cdot (i-1) \cdot \pi(C_{i-1}) + \pi(C_i) + \sum_{C_j^{(p)} \in S'} j \cdot \pi(C_j) \\
&\leq \ell \cdot (i-1) \cdot \pi(C_{i-1}) + \pi(C_i) + (i-2) \sum_{C_j^{(p)} \in S'} \pi(C_j) \\
&\leq \ell \cdot (i-1) \cdot \pi(C_{i-1}) + \pi(C_i) + (i-2)\left[\pi(C_i) - \ell \cdot \pi(C_{i-1})\right] \\
&= \ell \cdot \pi(C_{i-1}) + (i-1) \cdot \pi(C_i) \\
&\leq \frac{\pi(C_i)}{\pi(C_{i-1})} \cdot \pi(C_{i-1}) + (i-1) \cdot \pi(C_i) \\
&= \pi(C_i) + (i-1) \cdot \pi(C_i) = i \cdot \pi(C_i)
\end{aligned}
$$

□

With these results, we can derive the competitive ratio. According to Lemma 3, for each contract $C_i^{(j)} \in \mathcal{C}_{\leq t}(\text{Off2D})$, the accumulated cost $\Pi(C_i^{(j)})$ is bounded

by $i \cdot \pi(C_i)$. Therefore, summing up all the accumulated costs of each contract in $\mathcal{C}_{\leq t}(\text{OFF2D})$, we get the total cost of ON2D at time t. Note that every contract bought by ON2D must be totally covered by contracts in $\mathcal{C}_t(\text{OFF2D})$, since $\mathcal{C}_t(\text{OFF2D})$ is an optimal solution for the entire demand sequence $\sigma_{\leq t}$ and the contract independence property holds. Since we have k different contracts and for each contract C_i in $\mathcal{C}_t(\text{OFF2D})$, we have $\Pi(C_i) \leq i \cdot \pi(C_i) \leq k \cdot \pi(C_i)$, and:

Theorem 4. ON2D *is k-competitive, where k is the total number of contracts.*

As we will show in Section 3.5, this is almost optimal.

Finally, observe that restricting ON2D to Assumption A3 does not come at a large cost.

Theorem 5. *Let* ALG_1 *be an optimal offline algorithm for* PPP^2*, and let* ALG_2 *be an optimal offline algorithm for* PPP^2 *where we relax Assumption A3. Then* $\pi(\text{ALG}_2) \leq \pi(\text{ALG}_1) \leq 2 \cdot \pi(\text{ALG}_2)$.

Proof. Consider any contract $C_i^{(m)}(r_i, d_i)$ bought by an optimal offline algorithm for PPP^2 without Assumption A3. When time is divided into intervals of length d_i, $C_i^{(m)}$ will overlap in time at most two contracts $C_i^{(j)}$ and $C_i^{(l)}$ of duration d_i. Therefore, we can modify the optimal solution output by ALG_2 by purchasing those two contracts instead of $C_i^{(m)}$, eventually transforming the optimal solution output by ALG_2 into a feasible solution for PPP^2 (under Assumption A3). Hence, we can guarantee that $\pi(\text{ALG}_2) \leq \pi(\text{ALG}_1) \leq 2 \cdot \pi(\text{ALG}_2)$. □

Hence, since ON2D is k-competitive under Assumption A3 (Theorem 4), and since the optimal offline cost is at most a factor of two lower without the interval model (Theorem 3.4.3), we have:

Corollary 1. On2D *is* $2k$*-competitive for the general* PPP^2 *problem without Assumption A3, where* k *is the number of contracts.*

3.5 Lower Bound

Theorem 4 is essentially the best we can hope for:

Theorem 6. *No deterministic online algorithm can achieve a competitive ratio less than* $k/3$*.*

The proof is the 2-dimensional analogon of the proof in [31]. We consider a scenario where the next larger available contract doubles in cost. With k being the number of different contracts, each contract is $2k$ times longer and has $2k$ times more rate, i.e., in our plane representation contracts are squares covering an area $(2k)^{2i}$.

$$\pi(C_i) = 2^{i-1}$$
$$r_1 = 1; r_i = 2k \cdot r_{i-1} = (2k)^{i-1}$$
$$d_1 = 1; d_i = 2k \cdot d_{i-1} = (2k)^{i-1}$$

In the following, let us focus on a simple demand which only assumes rates $\sigma_t \in \{0, 1\}$ for all t. We let the adversary schedule demand only when On has no valid contract. For each interval $(2k)^i$ where the adversary asks for a 1-demand, On can choose between three options (see also Figure 3.4):

1 Eventual purchase of contract C_i. Assume that this happens n_i times.

2 Eventual purchase of larger contracts $C_j, j > i$. Assume that this happens $\sum_{j>i}^{k} n_j$ times.

3 Never purchase contract C_i or any larger contracts. Assume this happens m_i times.

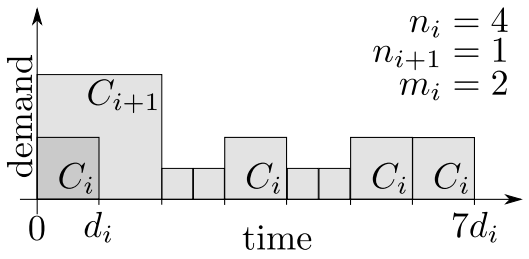


Figure 3.4: On Buys $n_i = 4$ Contracts C_i and $n_{i+1} = 1$ Contract C_{i+1} over Seven Intervals of Length d_i. In Two of These Seven Intervals On Buys Several Contracts Smaller than C_i to Cover the Demand.

Thus the sum of all contracts bought by On is given by $\pi(\text{On}) = \sum_{i=1}^{k} n_i \cdot \pi(C_i)$. Given an interval of length ℓ, we estimate the cost of Off by less than buying multiples of only one kind of contract over the full interval, i.e., ℓ/d_i contracts *for any* i: $\pi(\text{Off}) \leq \pi(C_i)(m_i + \sum_{j \geq i}^{k} n_j)$. In order to derive the lower bound we first prove a minimum cost of any algorithm On on intervals that start with a demand rate of 1.

Lemma 4. *Any* On *must pay at least* $\pi(C_i)$ *on each interval of length* d_i *that starts with a demand rate of* 1.

Proof. By induction on the different intervals 2^{i-1}. For $i = 1$, each algorithm must at least buy a contract of type C_1 in order to cover that demand. Assume that for $i-1$, it holds and now let us argue for i. If On does not buy a contract of type C_i, we can divide the volume into $(2k)^2$ squares with side length d_{i-1} each, where $2k \cdot d_{i-1} = d_i$. We let each of these $2k$ intervals (at the bottom row) start with a demand of 1 which then cost at least $\pi(C_{i-1})$ due to the induction hypothesis. The total cost is at least $2k \cdot \pi(C_{i-1}) = k \cdot \pi(C_i)$ for every interval where On does not buy a contract i and at least $\pi(C_i)$ otherwise.

□

Consider now an interval of length $(2k)^{i-1}$ where no contract of type i or higher was bought. We know from the induction that $\pi(\text{On}) \geq m_i \cdot k \cdot \pi(C_i)$. We can derive the following lower bound:

$$k \cdot \pi(\text{Off}) \leq \sum_{i=0}^{k} \left[\pi(C_i)(m_i + \sum_{j \geq i}^{k} n_j) \right] \tag{3.1}$$

$$\leq \sum_{i=0}^{k} \left[n_i \sum_{j=1}^{i} \pi(C_i) + m_i \cdot \pi(C_i) \right] \tag{3.2}$$

$$\leq \sum_{i=0}^{k} \left[2n_i \cdot \pi(C_i) + m_i \cdot \pi(C_i) \right] \tag{3.3}$$

$$\leq 3 \cdot \pi(\text{On}) \tag{3.4}$$

Inequality (3.1) is given by the cost estimation of Off against any On buying only one kind of contracts. Inequality (3.2) is a reorganization of the sum since $\pi(C_i)$ is multiplied by every $n_j, j \geq i$ which is also given after the reordering. Afterwards, we use the geometric sum on the cost of the contracts to derive Inequality (3.3). This leads to a lower bound of $k/3$ since $\pi(\text{On}) = \sum_{i=1}^{k} n_i \cdot \pi(C_i)$ and $\pi(\text{On}) \geq m_i \cdot k \cdot \pi(C_i)$.

3.6 Optimal Offline Algorithm

So far, we have treated the optimal offline algorithm on which On2D relies as a black box. In the following, we show that offline solutions can indeed be computed in polynomial time, and present a dynamic programming algorithm Off2D.

The basic idea behind the offline algorithm Off2D is that the optimal cost for any contract over a certain interval is obtained either by splitting the cost at some time, or by buying a long contract with a certain rate r. In the following, recall that d_k is the duration of the largest contract C_k.

Off2D proceeds as follows: It splits time into intervals of length d_k and solves each of these interval separately. Off2D relies on the following data structures: For

each d_k-length time interval I, we precompute the maximum demand within any subinterval $[i, j]$ of I, and store this information in position $M[i, j]$ of a $d_k \times d_k$ matrix M (Algorithm 4). In particular the maximum requested demand $\hat{\sigma}$ in interval I is stored in $M[1, d_k]$. A $d_k \times d_k \times \hat{\sigma}$ matrix `OP` is used to compute the optimal cost. The entry $\texttt{OP}[i, j, \lambda]$ indicates the optimal cost of covering a demand rate of $M[i, j] - \lambda$ over the interval $[i, j]$ — i.e. λ indicates the amount of *covered demand* for $[i, j]$. Initially, all entries are set to 0.

Algorithm 4 Pre-computation of matrix M

Input: Demand sequence $\sigma_t, \ldots, \sigma_{t+d_k}$ (over interval $[t, t + d_k]$).

Output: Matrix M.

1: **for** $i = 1$ to d_k **do**
2: $\quad M[i, i] \leftarrow \sigma_{t+i}$
3: **for** $i = 1$ to $d_k - 1$ **do**
4: $\quad$ **for** $j = i + 1$ to d_k **do**
5: $\quad\quad M[i, j] \leftarrow \max\{M[i, j - 1], \sigma_{t+j}\}$
6: **return** M

Algorithm 4 pre-computes the matrix M over the d_k-length interval $[t, t + d_k]$, where $t = b \cdot d_k$, for integer $b \geq 0$. Lines 1-2 initialize the matrix and store the demand σ_{t+i} in entry $M[i, i]$. Lines 3-5 compute the maximum demand within any time interval $[t+i, t+j]$, $0 \leq i \leq j \leq d_k$. The demand can be obtained by comparing the demand at time $t + j$ (i.e., σ_{t+j}) with the maximum demand between time $t + i$ and $t + j - 1$, which has already been computed by our algorithm.

After obtaining the matrix M over interval $[t, t+d_k]$, we can compute the optimal solution for PPP^2 problem over the same interval using Algorithm 5.

For the sake of ease of explanation, we assume that $t = 0$ in the arguments that follows. Off2D computes the optimal cost for any interval $[i, j]$ by either combining

Algorithm 5 Offline Algorithm OFF2D

Input: Precomputed matrix M over interval $[t, t + d_k]$.

Output: Optimal total costs $\texttt{OP}$ for all intervals within $[t, t + d_k]$.

Initialize all entries in $\texttt{OP}$ to be 0.
for $i = 1$ to d_k **do**
  for $\lambda = M[i, i] - 1$ to 0 **do**
    $\texttt{OP}[i, i, \lambda] \leftarrow \min_{C(r,d)\in\mathcal{C}}\{\texttt{OP}[i, i, \min\{\hat{\sigma}, \lambda + r\}] + \pi(r, d)$
for $\ell = 2$ to d_k **do**
  for $i = 1$ to $d_k - \ell + 1$ **do**
    $j = i + \ell - 1$
    for $\lambda = M[i, j] - 1$ to 0 **do**
      $\texttt{OP}[i, j, \lambda] \leftarrow \min_{i\le z<j}\{\texttt{OP}[i, z, \lambda] + \texttt{OP}[z + 1,\ j, \lambda]\}$
      $\mathcal{C}' \leftarrow \{C^{(x)}(t^{(x)}, r, d) \in \mathcal{C} : t^{(x)} = b \cdot d$ for some positive integer b and $t^{(x)} \le i < j \le t^{(x)} + d\}$
      if $\mathcal{C}'$ is not empty **then**
        $\texttt{OP}[i, j, \lambda] \leftarrow \min\{\texttt{OP}[i, j, \lambda];\ \min_{C(r,d)\in\mathcal{C}'} \texttt{OP}[i, j, \min\{\hat{\sigma}, \lambda + r\}] + \pi(r, d)\}$
return $\texttt{OP}[1, d_k, 0]$

the independent optimal solutions obtained for the subintervals $[i, z]$ and $[z+1, j]$, or by buying a long contract that covers a demand of r over the entire interval $[i, j]$. Note that according to Assumptions A1 and A2, these are the only two necessary options to consider. In the former case, it is obvious that the best splitting of the interval will give the minimum cost. In the latter case, a long contract with certain rate r is bought, however, the rate r may be lower than the maximum demand, $M[i, j]$, over $[i, j]$: In this case, we need to take into account how to best cover any remaining uncovered demand in $[i, j]$, which has been computed in $\texttt{OP}[i, j, \lambda + r]$.

Theorem 7. OFF2D *computes an optimal offline solution for any given interval of length d_k in time $O(d_k^3 \cdot \widehat{\sigma})$, where $\widehat{\sigma}$ is the maximum demand over the interval.*

Proof. We again assume for the sake of simplicity and without loss of generality, that $t = 0$ and the d_k-length interval we consider is $[0, d_k]$.

Correctness: By induction over the length of the subintervals $\ell = j - i + 1$ and the respective uncovered demand λ. Clearly, the claim is true for intervals $[i, i]$ ($\ell = 1$) (Lines 2-4): If $\lambda > 0$ we need at least one contract $C(r, d)$ to finish covering the demand at time i; the remaining demand at time i not covered by C must be covered optimally by other contracts, as previously computed in $\texttt{OP}[i, i, \lambda + r]$.

For any $\texttt{OP}[i, i, \lambda]$, we compute the optimal remaining cost when λ demand is already covered. In order to obtain this value, we need to buy a new contract $C(r, d)$ which covers r units if demand of the $M[i, j] - \lambda$ uncovered untis of demand. Here any contract $C(r, d) \in \mathcal{C}$ should be considered. After the newly bought contract, $\lambda + r$ demand has already been covered and the cost to cover the remaining $M[i, j] - (\lambda + r)$ has already been computed and stored in $\texttt{OP}[i, i, \lambda + r]$. Hence, summing up $\texttt{OP}[i, i, \lambda + r]$ and the cost $\pi(r, d)$ for the new contract $C(r, d)$, we get the value for $\texttt{OP}[i, i, \lambda]$.

Now consider a subinterval $[i, j]$ of length $\ell = j - i + 1 \geq 2$, where $1 \leq i \leq j \leq d_k$. This interval is either split into two non-overlapping subintervals of smaller length (Case I), or a long contract of length equal to or greater than ℓ that completely covers $[i, j]$ is bought, at a certain demand rate r, where $0 \leq r \leq M[i, j]$ (Case II). Given Assumption A2 and A3, for any instances of contracts $C_x^{(y)}$ and $C_p^{(q)}$, *either the duration of one contract is fully contained in the other*, or the *two contracts never overlap in time*: Hence, given that we consider all intervals $[i, j]$, including the ones that may correspond to actual instances of contracts, it is enough to consider only these two cases.

In Case I, we split the interval at time z such that the solution $\texttt{OP}[i, z, \lambda] + \texttt{OP}[z + 1, j, \lambda]$ is minimized over all z between i and j (Line 9). Since the lengths of the two subintervals $z - i + 1$ and $j - z$ are both smaller than ℓ, $\texttt{OP}[i, z, \lambda]$ and $\texttt{OP}[z + 1, j, \lambda]$ already store the cost of optimal solutions for these subproblems, respectively, by the induction hypothesis. Hence $\texttt{OP}[i, z, \lambda] + \texttt{OP}[z + 1, j, \lambda]$ will yield the optimal solution for $\texttt{OP}[i, j, \lambda]$ if Case I applies.

In Case II, we buy a long contract with rate r. First, we need to check which contracts with longer durations can cover $[i, j]$ fully, and store the candidate contracts in $\mathcal{C}'$. A candidate contract $C^{(x)}(t^{(x)}, r, d)$, where $t^{(x)} = b \cdot d$ according to Assumption A3, satisfies $t^{(x)} \leq i < j < t^{(x)} + d)$. The algorithm picks the valid candidate contract that minimizes $\pi(r, d)$ plus the optimal cost of covering the largest remaining demand $M[i, j] - (\lambda + r)$ over $[i, j]$, which has been previously computed and stored in $\texttt{OP}[i, j, \lambda + r]$ (Line 10).

By choosing the smaller value of Cases I and II, we obtain the optimal cost for subproblem $[i, j, \lambda]$ (Line 12).

Time Complexity: Now we consider the time complexity of Off2D. The total time complexity for the pre-computation part in Algorithm 4 is $O(d_k^2)$. The first part of Algorithm 5 in (Lines 2-4) takes $O(d_k \cdot k \cdot \widehat{\sigma})$ time, where $\widehat{\sigma}$ is the maximum demand for the whole time interval. The first two loops of the second part (Lines 5-6) take $O(d_k^2)$ time and the for-loop in Line 8 takes $O(\widehat{\sigma})$ time. The statement in Line 9 requires $O(d_k)$ time and Lines 10 and 12 take time $O(k)$ each. Therefore, the total time complexity is $O(d_k^3 \cdot \widehat{\sigma})$ for a subinterval with length d_k.

Thus, by summing up the computation time of $\lceil |\sigma|/d_k \rceil$ subintervals of length d_k, where $|\sigma|$ is the total duration of σ, we have an overall complexity of $O(|\sigma| \cdot d_k^2 \cdot \widehat{\sigma})$.

□

3.7 Higher Dimensions

OFF2D and ON2D are designed for the two-dimensional version of the PPP problem but they can also be extended towards a D-dimensional version of the problem, where each additional dimension (other than the time duration dimension) would indicate the rate at which you would buy a certain resource. Regarding OFF2D we need to do the following changes: For each additional dimension we need to extend the dimension of the optimal cost matrix `OP` by one and add two additional loops in OFF2D's Algorithm 5. Furthermore we only need to add one additional dimension for the M matrix in Algorithm 4 which indicates the current demand dimension β, $M[i, j, \beta]$ and run the algorithm β times for the pre-computation.

Assume a scenario where a third dimension is added, e.g. computational and network resources over time. The contracts $C(r, r', d)$ then cover $r \times r' \times d$ cuboids. In order to adjust Algorithm 5, we add another loop after Line 3 which goes through the maximum values λ' of the additional demand (**for** $\lambda' = M[i, i, 2] - 1$ to 0 **do**) and change the statement in Line 4 to: $\texttt{OP}[i, i, \lambda, \lambda'] \leftarrow \min_{C(r,r',d)\in\mathcal{C}'} \texttt{OP}[i, i, \lambda + r, \lambda' + r'] + \pi(r, r', d)$. The same loop must also be added after Line 8 and the updates of the `OP` matrix must be changed accordingly in Lines 9 and 12.

The runtime of the pre-computation in Algorithm 4 would be increased by a factor of D (i.e., by the dimension of the problem) and still be negligible regarding the overall runtime (assuming D is a constant). For Algorithm 5 the runtime would increase by a factor of $\Pi_{i\geq 2}\widehat{\sigma}^i$, where σ^i is the maximum demand for resource i, for $i \geq 1$, leading to an overall runtime of $O(d_k^3 \cdot \Pi_{i\geq 1}\widehat{\sigma}^i)$ for each interval d_k.

No changes are needed regarding ON2D. It still mimics OFF2D's behavior and given the Assumptions A2 and A3, the contract independence still holds for higher dimensions. Hence, the proof for the competitive ratio of k still applies.

The offline algorithm, for the D-dimensional problem is presented in Algorithms 6 and 7. As in two dimensions, the D-dimensional *online algorithm* can simply mimick the D-dimensional offline algorithm. Given the resulting contract independencies, the $O(k)$-competitiveness is maintained.

Algorithm 6 Pre-computation of matrix M

Input: Demand sequences $\sigma_t^{dim}, \ldots, \sigma_{t+d_k}^{dim}$ (over interval $[t, t+d_k]$) with $dim \in \{1, 2, ..., D-1\}$.

Output: Matrix M.

1: **for** $i = 1$ to d_k **do**
2: **for** $dim = 1$ to $D-1$ **do**
3: $M[i, i, dim] \leftarrow \sigma_{t+i}^{dim}$
4: **for** $i = 1$ to $d_k - 1$ **do**
5: **for** $j = i+1$ to d_k **do**
6: **for** $dim = 1$ to $D-1$ **do**
7: $M[i, j, dim] \leftarrow \max\{M[i, j-1, dim], \sigma_{t+j}^{dim}\}$
8: **return** M

Algorithm 7 Offline Algorithm OFFD-D

Input: Precomputed matrix M over interval $[t, t+d_k, D]$.

Output: Optimal total costs $\texttt{OP}[i, j, \cdot, \cdot, \ldots, \cdot]$ for all intervals within $[t, t+d_k]$.

Initialize all entries in $\texttt{OP}$ to be 0 and $dim = D-1$.
for $i = 1$ to d_k **do**
  for $\lambda_1 = M[i,i,1] - 1$ to 0 **do**
    for $\lambda_2 = M[i,i,2] - 1$ to 0 **do**
      $\vdots$
      for $\lambda_{dim} = M[i,i,dim] - 1$ to 0 **do**
        $\texttt{OP}[i,i,\lambda_1,\lambda_2,\ldots,\lambda_{dim}] \leftarrow \min_{C(r_1,r_2,\ldots,r_{dim},d)\in\mathcal{C}} \texttt{OP}[i,i,\lambda_1 + r_1, \lambda_2 + r_2, \ldots, \lambda_{dim}] + \pi(r_1, r_2, \ldots, r_{dim}, d)$
for $\ell = 2$ to d_k **do**
  for $i = 1$ to $d_k - \ell + 1$ **do**
    $j = i + \ell - 1$
    for $\lambda_1 = M[i,j,1] - 1$ to 0 **do**
      for $\lambda_2 = M[i,j,2] - 1$ to 0 **do**
        $\vdots$
        for $\lambda_{dim} = M[i,j,dim] - 1$ to 0 **do**
          $\texttt{OP}[i,j,\lambda_1,\lambda_2,\ldots,\lambda_{dim}] \leftarrow \min_{i\le z<j}\{\texttt{OP}[i,z,\lambda_1,\lambda_2,\ldots,\lambda_{dim}] + \texttt{OP}[z+1,j,\lambda_1,\lambda_2,\ldots,\lambda_{dim}]\}$
          $\mathcal{C}' \leftarrow \{C^{(x)}(t^{(x)}, r_1, r_2, \ldots, r_{dim}, d) \in \mathcal{C} : t^{(x)} = b \cdot d$ for some positive integer b and $t^{(x)} \le i < j \le t^{(x)} + d\}$.
          if $\mathcal{C}'$ is not empty **then**
            $\texttt{OP}[i,j,\lambda_1,\lambda_2,\ldots,\lambda_{dim}] \leftarrow \min\{\texttt{OP}[i,j,\lambda_1,\lambda_2,\ldots,\lambda_{dim}]; \min_{C(r_1,r_2,\ldots,r_{dim},d)\in\mathcal{C}'_1} \texttt{OP}[i,j,\lambda_1 + r_1, \lambda_2 + r_2, \ldots, \lambda_{dim} + r_{dim}] + \pi(r_1, r_2, \ldots, r_{dim}, d)\}$
return $\texttt{OP}[1, d_k, 0, 0, \ldots, 0]$

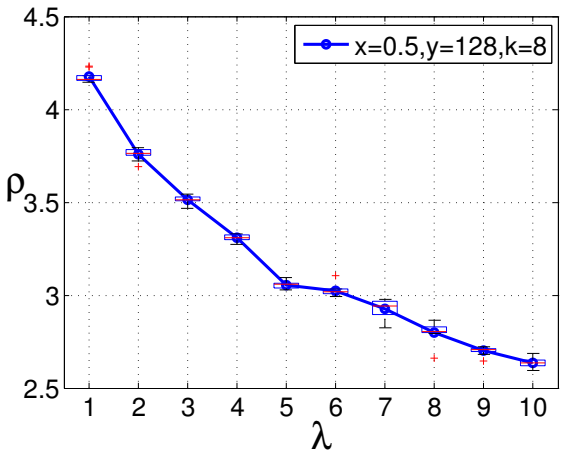
3.8 Simulation

We have conducted a small simulation study to complement our formal analysis. In this simulation, we consider k different square contracts where each contract $C_i(r_i, d_i)$ has rate and duration $r_i = d_i = 2^{i-1}$, for $1 \leq i \leq k$. The price π of a contract is a function of the rate-duration product $r_i \cdot d_i$, and we study a parameter x to vary the discount. Concretely, we consider a scenario where a twice as large time-rate product is by factor $(1 + x)$ more expensive, i.e., $\pi(2 \cdot d \cdot r) = (1 + x) \cdot \pi(d \cdot r)$; we set $\pi(1) = 1$.

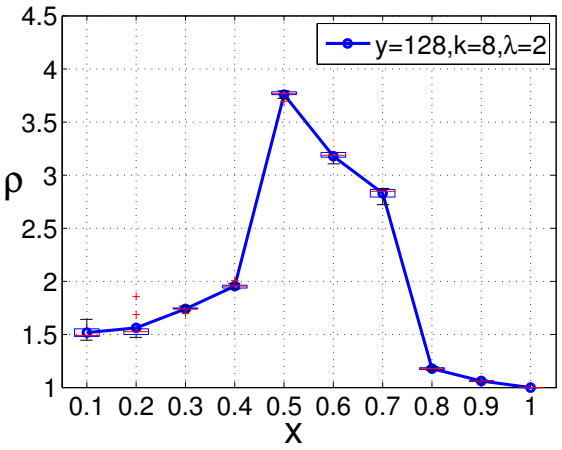
To generate the demand σ, we use a randomized scheme: non-zero demand requests arrive according to a Poisson distribution with parameter λ, i.e., the time between non-zero σ_t is exponentially distributed. For each non-zero request, we sample a demand value uniformly at random from the interval $[1, y]$.

In this experiment, each simulation run represents 1000 time steps, and is repeated 10 times.

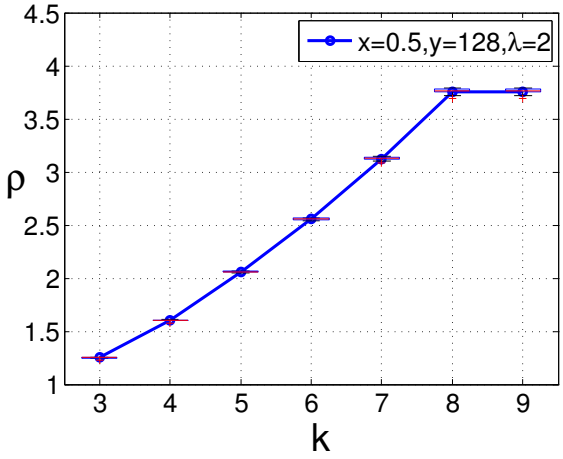
Impact of the request model. We first study how the competitive ratio depends on the demand arrival pattern. Figure 3.5a plots the competitive ratio ρ as a function of the Poisson distribution parameter λ. The price model with $x = 0.5$ is used, and there are $k = 8$ contract types. First, we observe that the competitive ratio ρ is bounded by approx. 5, which is slightly lower than what we expect in the worst-case (cf Theorem 4). Another observation is that the competitive ratio decreases as λ increases. This can be explained by the fact that demand rates become sparser for increasing λ, and hence less contracts will be bought. Meanwhile, when the demand rates are sparse, the offline algorithm will have less chance to buy a larger contract. Put differently, the online algorithm will pay relatively more compared to the offline algorithm for small λ, as it purchases more small contracts.



(a) Effect of Request Distribution (Poisson λ)



(b) Effect of Discount x



(c) Effect of The Number of Contracts k

Figure 3.5: Simulation Results under Different Settings.

Impact of the price model. Different price models also affect the purchasing behavior.Figure 3.5b shows the competitive ratio ρ for different values x. (For this scenario, we set $y = 128$, $k = 8$ and $\lambda = 2$.) We see a tradeoff: for small x, until $x = 0.5$, the competitive ratio increases and then begins to decrease again. The general trend can be explained by the fact that for small x, it is worthwhile to buy larger contracts earlier, and it is hence impossible to charge On2D much; for larger x, also an offline solution cannot profit from buying a large contract.

Impact of the number of contracts. Finally, Figure 3.5c shows the competitive ratio as a function of the number of contracts k. (We fix $x = 0.5$, $y = 128$ and $\lambda = 2$ in this simulation.) The competitive ratio first increases as k increases but then stabilizes. This stabilization is due to the fact that when we have eight or more contracts ($k \geq 8$), the largest contract can cover the maximum rate. In the beginning, the ratio increases since the offline algorithm buys larger and larger contracts, and the online algorithm pays for many small contracts along the way.

3.9 Conclusion

This work has shown that at the heart of efficient cloud resource allocation lies a fundamental algorithmic problem, and we introduced the PPP^2 problem, a *2-dimensional* variant of the *online Parking Permit Problem* PPP. We presented a deterministic online algorithm On2D that provably achieves a competitive ratio of k, where k is the total number of available contracts; if we relax Assumption A3, the competitive ratio of our algorithm is $2k$. We also showed that On2D is almost optimal in the sense that no deterministic online algorithm for PPP^2 can achieve a competitive ratio lower than $k/3$. Finally, we proved that the offline version of PPP^2 can be solved in polynomial time.

We believe that our work opens interesting directions for future research.

1. *Optimality:* The obvious open question regards the gap between the upper bound k and the lower bound $k/3$ derived in this work.

2. *Interval assumption:* Interestingly, while the interval model only comes at the cost of a small constant approximation factor, it seems hard to remove the restriction while keeping optimality. In fact, we conjecture that computing an optimal offline solution may even be NP-hard in general.

3. *Randomized algorithms:* It will be interesting to study whether the randomized algorithms known from the classic parking permit problem can also be generalized to multiple dimensions.

Chapter 4

HOW TO BUY DISCOUNT VOUCHERS WITH EXPIRATION DATES? A COMPETITIVE ANALYSIS APPROACH

4.1 Introduction

In our modern times where alternatives are plenty, companies constantly look for new ways to differentiate themselves from competitors, offering a wide range of complex discounts and vouchers. For example, many gyms today offer daily, weekly and yearly passes, at different rates. Also other consumable good industries offer discount vouchers which expire independently of whether they have been used up by then or not. Companies such as Groupon even follow a business model which to a large extent consists in offering various discount vouchers for other companies.

Whether it makes sense to buy a pass or discount voucher with expiration date is a difficult decision problem, due to uncertainty: customers often do not know how frequently and how long they will be interested in a certain consumable good or service. In the worst-case, right after having bought a yearly pass or a large number of vouchers, a customer may lose interest or time.

We, in this work, initiate the study of the discount voucher purchasing problem: the problem of how to buy discount vouchers with expiration dates, in an online fashion, i.e., without knowing the future. We assume a conservative worst-case perspective and consider Murphy's Law: whether and when a customer can make use of its vouchers is determined by an adversary.

We are in the realm of online algorithms and competitive analysis, and we are interested in online voucher buying strategies which are competitive against an optimal

offline algorithm: the overall amount of money spent by the online algorithm is not much higher than the money spent by a clairvoyant optimal offline algorithm, which knows the entire future demand ahead of time.

The problem shares many similarities with evergreen online optimization problems such as the the Parking Permit Problem [31] and the Bahncard Problem [19], where benefits expire with time, regardless of usage. We consider the following two problem variants:

1. **Once-Expired-Lose-Discount (OELD):** The voucher loses its discount for the unused face value after its expiration date. However, its value can still be used to buy goods at their regular price.

2. **Once-Expired-Lose-Everything (OELE):** The voucher loses its entire value after its expiration date.

Sometimes, companies even limit the time window in which vouchers can be bought. We will refer to this variant by **Limited Purchasing Window (LPW)**.

Our Contribution. We initiate the study of the online voucher purchase problem, from a competitive analysis perspective. We present deterministic online algorithms to compute a 4-competitive solution for OELE—which is optimal due to the multi-discount ski rental lower bound [47]—and an 8-competitive solution to OELD. We also prove a lower bound on the LPW variant, showing that the problem is inherently hard. Additionally, we also consider offline versions of the problems, and present a polynomial-time optimal offline algorithm for OELE and LPW, as well as a 2-competitive solution for OELD.

Organization. The remainder of this work is organized as follows. In Section 4.2, we review related work. In Section 4.3, we introduce three variants of the discount

voucher purchasing problem and formulate our model. In Section 4.4, we discuss online algorithms and competitive analysis for each variant respectively. In Section 4.4.1, we present a case study and provide the general analysis for OELE. In Section 4.4.2, we discuss the competitive analysis for OELD using the same algorithm for OELE. In Section 4.4.3, we provide a lower bound analysis for LPW. In Section 4.5, we present algorithms for these three variants. We present a polynomial-time optimal offline algorithm for OELE in Section 4.5.1, and prove it to be be 2-competitive solution to OELD in Section 4.5.2 and an optimal solution to LPW in Section 4.5.3. Finally, we conclude our work in Section 4.6.

4.2 Related Work

Our work is motivated by the practical discount voucher purchasing problem and by the online purchasing manner, i.e., the future demands of customers are unknown in advance. One frequently used assumption in many online models is that once the resource has been purchased, it will never expire, such as ski-rental problems [11], facility location problems [21], buy-at-bulk [35] and rent-or-buy [28] problems. Some other online problems, such as the Parking Permit Problem [31] and the Bahncard Problem [19], instead assume purchases have time durations which expire regardless of whether the purchase is used or not. Our work works on the second model which assumes that the resource will expire after certain duration.

This work focuses on an online resource purchasing problem in which discount vouchers or groupons are considered as resource to be bought. This work is motivated by a real discount voucher or groupon purchasing problem. Refer to [44] and [43] respectively for general information. The topics which are close to our work are ski-rental problem or parking permit problem. The ski rental problem proposed in [27] is one of the most fundamental problems and the authors in [27] studied its

online version in economical aspects. There have been quite many generalizations to this problem. For example, the authors in [36] studied the situation in which the purchasing price varies over time while the rental cost stays at a fixed price. Ali et al. [3] introduced a variant of the problem by assuming the skier knows the first (or second) moment of the distribution of the number of ski days in a season. They showed that knowing this information leads to achieving the best worst-case expected competitive ratio performance. Other classic generalizations of this problem include the replacement problem of Chunlin et al. [15] and cloud resource management [41]. Recently, Meyerson [31] proposed the parking permit problem and Hiroshi et al. extends the classic ski rental problem to multislope ski rental problem [26].

The paper closest to our work is the ski rental problem studied by Guiqing et al. [47]. The authors proposed the ski-rental problem with multiple discount options in which there are n options each with a rental duration respectively. They presented a 4-competitive online algorithm for continuous request sequence scenarios and also showed a matching lower bound for this problem. Our work considers a discount voucher purchasing problem with expiration dates, in which the cost of a voucher is computed based on the volume of the resource, i.e., the number of yoga visits when the voucher is a yoga pass. In [47], the contract is computed based on the time duration. The first variant OELE studied in this work is similar to the ski rental problem except that vouchers will expire at expiration date. Therefore, OELE is multi-discount ski rental problem in each discount window. However, we extend this problem to another two variants (OELD and LPW) and show approximation analysis for each respectively. Our algorithm can work for any request sequences, while the requests are required to be continuous in [47], i.e., every day there must be a request. In addition to online algorithms with competitive analysis, we also make an effort on the offline algorithm.

4.3 Model

We are given a range of k different discount vouchers of increasing value $V = (v_1,\ v_2,\ \cdots,\ v_k)$. For simplicity, we will use v_i to denote both the voucher as well as its value, i.e., the number of consumable goods it represents. The price of the voucher v_i is denoted by $\Pi(v_i)$, and the price per unit consumable is denoted by $\pi(v_i)$, i.e., $\pi(v_i) = \Pi(v_i)/v_i$. Without loss of generality (w.l.o.g.), we assume that $\pi(v_j) \leq \pi(v_i)$ for $j > i$: more expensive, smaller vouchers are never interesting and can be ignored. We will denote by v_0 the "voucher" which represents a single unit, without discount; its price is denoted by $\pi(v_0) = \Pi(v_0)$.

Each discount voucher has an expiration date, which is independent of the consumption of the good. After expiring, a voucher v_i may either lose its entire value $\Pi(v_i)$, or it may fall back to its so-called *face value*: in this case, it can only be used to buy $\lfloor \Pi(v_i)/\Pi(v_0) \rfloor \leq v_i$ many goods. We will refer to the first variant by **Once-Expire-Lose-Everything (OELE)**, and to the second variant by **Once-Expire-Lose-Discount (OELD)**. Sometimes, we will also restrict the time window in which new vouchers can be bought: we will refer to this variant by **Limited Purchasing Window (LPW)**.

We model the demand for a given consumable good over time using a binary *consumption request sequence* $\sigma = (\sigma_1,\ \sigma_2,\ \ldots)$: the entry $\sigma_t = 1$ denotes that the good is consumed at time t, and $\sigma_t = 0$ denotes that the good is not consumed. If clear from the context, we will sometimes refer by σ_t to the time t of the request. All algorithms and results in our work can easily be extended if the demand for a good is non-binary. The sequence σ is revealed over time to the online algorithm: at time t, the online algorithm only knows the first $t-1$ entries, henceforth denoted by $\sigma_{<t}$. Whenever a consumption request occurs ($\sigma_t = 1$), the online algorithm either has to

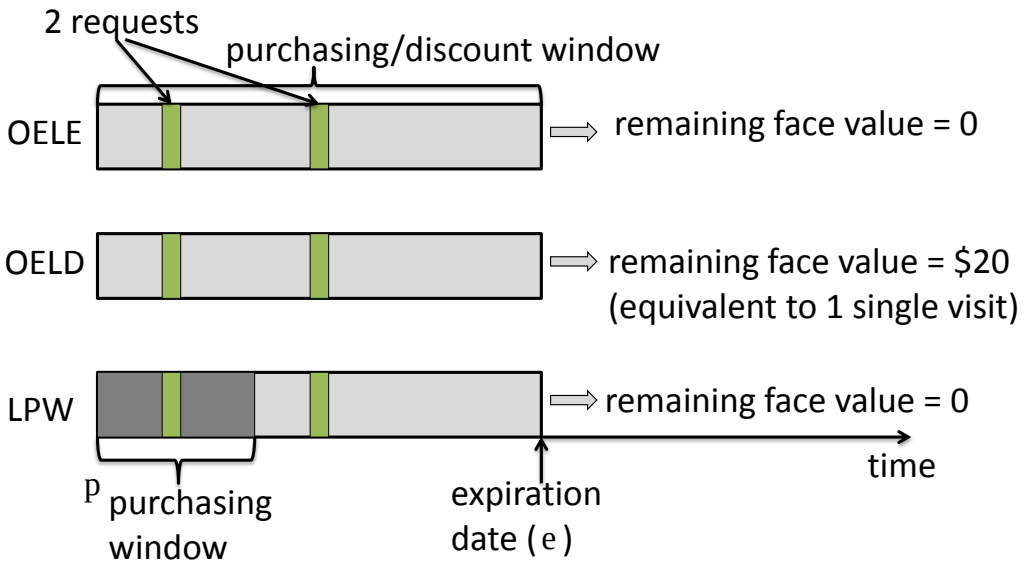


Figure 4.1: Example for OELE, OELD and LPW Given a Voucher $v_1 = 4$ with $\Pi(v_1) = 40$ and a Single-unit Option v_0 with Price $\Pi(v_0) = 20$. When v_1 Is Bought within the Purchasing Window, the Remaining Face Value after the Voucher Expires Is Shown for Each Variant.

have a voucher ready or buy a new one (possibly v_0). The offline algorithm always knows the entire sequence σ.

Example. Figure 4.1 gives an example to describe our model in details. Consider a yoga voucher v_1 for 4 entries costs 40 dollars, i.e., $v_1 = 4$ and $\Pi(v_1) = 40$, while a single entry v_0 without discounts costs 20 dollars ($\Pi(v_0) = 20$). If Alice buys the voucher v_1 at her first visit, and ends up going only to two classes before the voucher expires, she will not have won anything in the OELE model compared to a scenario where she buys two v_0 single vouchers. However, in the OELD model, there will be 20 dollars face value left after the expiration date, which is worth one additional single visit. In the LPW model, there also exists the constraint that the purchasing time window is limited (marked as dark grey in the figure). Alice either purchases the

vouchers within this purchasing window, or she is forced to buy single entries (i.e., v_0's). $\square$

Throughout this work, we will make the practically motivated assumption that vouchers are issued for disjoint time intervals, and start at purchase time p and expire at time e. That is, using the notation p_i and e_i for voucher v_i, we have for any two vouchers v_j and v_ℓ we assume $p_j \leq e_j < p_\ell \leq e_\ell$ or $p_\ell \leq e_\ell < p_j \leq e_j$.

Let ALG be an algorithm and let $\Pi_{\text{ALG}}(\sigma)$ denote the overall cost that ALG incurs on a request sequence σ. Moreover, let $\Pi_{\text{ALG}}(\sigma_{\leq t})$ denote the cost of ALG from the beginning of σ through time t.

Our goal is to find an online algorithm ON whose cost is not much larger than the cost of an optimal offline algorithm OFF. Concretely, we seek to minimize the (worst-case) competitive ratio ρ over all σ: $\rho = \max_\sigma(\Pi_{\text{ON}}(\sigma)/\Pi_{\text{OFF}}(\sigma)) + c$, where c is some additive constant.

4.4 Competitive Online Algorithm

This section first studies the OELE and OELD problem variants and presents competitive online algorithms in turn, together with an analysis. Subsequently, we prove that LPW is inherently hard to solve in an online setting.

4.4.1 Once-Expired-Lose-Everything (OELE)

The OELE problem variant can be seen as a generalization of classic ski rental: the ski rental problem can be modeled using two contracts v_0 and v_1, where v_1 has an infinite duration and does not expire: the skis are bought. It is also a generalization of multi-discount ski rental [47] as in each discount window each voucher in OELE can be seen as a contract in multi-discount ski rental problem. The volume of the voucher can be seen as the duration of the contract. The expiration date can be set

as the date for the last request. After the vouchers are expired and the next discount window starts, we apply a new generalization of multi-discount ski rental problem. Therefore, our problem is volume based in which each voucher has an expiration date while multi-discount ski rental problem is time based. This variant is provided as a building block for the other variants, OELD and LPW, for both online and offline versions.

The Algorithm

In order to solve this problem, we propose the following algorithm On, see Algorithm 8. For ease of explanation, assume dummy vouchers v_{-1} and v_{k+1} with $\Pi(v_{-1}) = 0$ and $\Pi(v_{k+1}) = \infty$. On pursues an amortization strategy: it always seeks to buy the largest voucher whose cost is less than or equal to the accumulated cost within the current discount window.

On maintains two values: C to store the total accumulated cost and C' to indicate the cost incurred within each discount window. We denote by $\tau_1,\ \tau_2, \cdots$ the time point at which On decides to buy a new voucher, i.e., τ_i indicates the time for the first request covered by the ith voucher. We use j as an index to indicate the current request (i.e., the jth request occurs at time σ_j), and i to indicate the index of the current voucher that On is going to buy. τ_s indicates the starting point of the requests for sth voucher bought by our online algorithm, where s is an index starting from 1. Initially, C is set to be 0, while j and s are both set to be 1 (Line 1). In the beginning of each discount window, C' is initially set to be 0 and i is set to be 1 (Line 3). Therefore, we can get the starting point τ_s of each bought voucher which is denoted by σ_j (Line 4). Then we can update the current discount window. For example, if the expiration date is t, the last request is σ_e where $\sigma_e \leq t$ and σ_e is the largest time which is less than or equal to t. Hence, the current discount window spans from σ_j to

Algorithm 8 Online algorithm On

Input: online request sequence $\sigma = (\sigma_1, \sigma_2, \ldots)$

Output: overall cost

1: $C \leftarrow 0, j \leftarrow 1, s \leftarrow 1$
2: **while** new $\exists$ request σ_j **do**
3: $C' \leftarrow 0, i \leftarrow 0$
4: $\tau_s \leftarrow \sigma_j$
5: **update** discount window
6: **for** each new request at σ_j coming within the current discount window **do**
7: **if** request at σ_j is not yet covered by a voucher **then**
8: buy voucher v_i
9: **update** i s.t. $\Pi(v_i) \leq C' < \Pi(v_{i+1})$
10: $C' \leftarrow C' + \Pi(v_i)$
11: $j \leftarrow j + 1$
12: $s \leftarrow s + 1$
13: $C \leftarrow C + C'$
14: **return** C

σ_e. After the initialization of the data structures, On computes the cost C' for each discount window separately (Lines 6-Line 11). If the current request σ_j is not covered by a voucher, buy voucher v_i and update index i such that $\Pi(v_i) \leq C < \Pi(v_{i+1})$ (Lines 8-Line 9). In other words, On buys the largest voucher whose cost is less than or equal to the accumulated cost within the current discount window. Update the cost C' by adding voucher cost $\Pi(v_i)$ into it (Line 10). Increment request index j by 1 (Line 11). After finishing dealing with all the requests within the current discount window, increment index s by 1 (Line 12) and add the cost C' obtained within the current discount window to the total cost C (Line 13).

Case Study

In order to illustrate the behavior and the performance of On within a discount window in some specific scenarios, we consider the following study cases: let the vouchers and discount be the way such that there will always exist such voucher v_i satisfying $f(v_i) = 2^{i-1}$ for integer i where $1 \leq i \leq k$. Assume that $v_1 = v_0$ is a single visit.

Now let us take a look at how On behaves in the following two study cases.

Case 1: We assume that the request sequence T is not sufficient enough for On such that On cannot buy multiple v_k's and the largest voucher On purchase is v_x, where $1 \leq x \leq k$. It is trivial to see that initially On will purchase v_1 with cost 1 and another v_1 for the second request. After that, On will turn to buy each voucher from v_2 to v_i once sequentially. Hence, we have:

$$\begin{aligned}\Pi(\text{On}) &= \Pi(v_1) + \Pi(v_1) + \Pi(v_2) + \cdots + \Pi(v_x) \\ &= 1 + 1 + 2 + \cdots + 2^{x-1} \\ &= 1 + \sum_{i=0}^{x-1} 2^i \\ &= 2^x = 4 \cdot 2^{x-2}\end{aligned}$$

Let the total number of requests be V. Note that except for the last voucher, the length of the voucher is the number of the requests within this voucher. Therefore, we can obtain V by summing up all the vouchers before v_x and the real data requests within the last voucher v_x. In the following, we use $\Pi^{-1}(c)$ to denote the length of the voucher whose cost is equal to c.

$$
\begin{aligned}
Vol &> \Pi^{-1}(1) + \Pi^{-1}(1) + \Pi^{-1}(2) + \cdots + \Pi^{-1}(2^{x-2}) \\
&= \Pi^{-1}(1) + \sum_{i=0}^{k-2} \Pi^{-1}(2^i) \\
&> \Pi^{-1}(2^{x-2})
\end{aligned}
$$

Therefore, we have $\Pi(\textsc{Off}) \geq \Pi(Vol) > \Pi(f^{-1}(2^{k-2})) = 2^{x-2}$ and hence $\rho = \frac{\Pi(\textsc{On})}{\Pi(\textsc{Off})} < 4$.

Case 2: In this case we assume that On buys multiple vouchers v_k, i.e., On buys α vouchers v_k where $\alpha \geq 2$. Similarly On will buy 2 vouchers v_1 and then turn to buy each voucher from v_2 to v_{k-1} once sequently. Finally On keeps purchasing the same voucher v_k for α times. We can compute the cost of the online algorithm and offline algorithm similarly:

$$
\begin{aligned}
\Pi(ON) &= \pi(v_1) + \pi(v_1) + \pi(v_2) + \cdots + \alpha \cdot \pi(v_k) \\
&= 1 + 1 + 2 + \cdots + \alpha \cdot 2^{k-1} \\
&= 1 + \sum_{i=0}^{k-1} 2^i + (\alpha - 1)2^{k-1} \\
&= \frac{\alpha + 1}{2} 2^k = (\alpha + 1) \cdot 2^{k-1}
\end{aligned}
$$

Now we compute the total requests Vol:

$$
\begin{aligned}
Vol &> \pi^{-1}(1) + \pi^{-1}(1) + \pi^{-1}(2) + \cdots + \pi^{-1}(2^{k-2}) + (\alpha - 1)\pi^{-1}(2^{k-1}) \\
&= \pi^{-1}(1) + \sum_{i=0}^{k-2} \pi^{-1}(2^i) + (\alpha - 1)\pi^{-1}(2^{k-2}) \\
&> (\alpha - 1)\pi^{-1}(2^{k-1})
\end{aligned}
$$

It is trivial to see that within a discount window the cost of the offline algorithm should be at least equal to the total cost of $\alpha - 1$ vouchers v_k, i.e., $\Pi(\textsc{Off}) \geq (\alpha - 1)2^{k-1}$. Therefore, $\rho = \frac{\Pi(\textsc{On})}{\Pi(\textsc{Off})} < \frac{(\alpha+1)\cdot 2^{k-1}}{(\alpha-1)\cdot 2^{k-1}} \leq 2$. Since the behavior of On for

each discount window is independent for OELE, we can derive the competitive ratio for the whole request sequence σ is 4.

General Analysis

Note that $\tau_1,\ \tau_2, \cdots$ are the starting points of the vouchers at which On decides to buy a new voucher. To discuss the competitive analysis of our algorithm, let τ_0 be 0, which indicates that there is no voucher bought yet. Let the ith voucher bought be v'_i.

Now we prove that the competitive ratio $\rho = \Pi_\tau(\text{On})/\Pi_\tau(\text{Off}) \leq 4$ for all τ's within each discount window, where $\tau \in (\tau_1,\ \tau_2, \cdots)$ is the starting point of vouchers bought by On. Lemma 5 proves the competitive ratio for the requests within the first discount window. It is obvious to show the ratio for the requests within other discount windows by rebuilding the index i of starting point τ_i. For example, if the starting point sequence for a discount window starts at τ_x, we just need to set $\tau_j = \tau_{x+j-1}$, where $j \geq 1$. We discuss the ratio between the cost of On before it decides to buy a new voucher $\Pi_{\tau_{m-1}}(\text{On})$ (note that between vouchers, On will not buy any other vouchers) and the cost of Off after On decides to buy this new voucher is $\Pi_{\tau_m}(\text{Off})$.

Lemma 5. *The cost of* On *before it decides to buy a new voucher at* τ_m *is at most twice the cost of* Off *after* On *decides to buy this new voucher for request* τ_m*, i.e.,* $\Pi_{\tau_{m-1}}(\text{On})/\Pi_{\tau_m}(\text{Off}) \leq 2$ *within each discount window, where* $m \geq 1$.

Proof. We argue by induction on m[1]. Note that $\tau_0 = 0$.

Base case ($m = 1$ and $m = 2$): When $m = 1$, according to our algorithm On, $\Pi_{\tau_0}(\text{On}) = 0$. Since there is a request at time τ_1 and the offline algorithm will buy

[1]This work was developed concurrently and independently to [47]; however this proof shows many similarities to the one in [47].

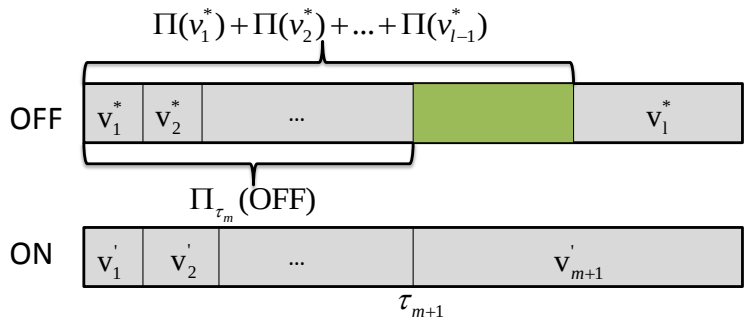
a voucher v_0, we have $\Pi_{\tau_1}(\text{OFF}) = \Pi(v_0)$. Therefore, we have $\Pi_{\tau_0}(\text{ON})/\Pi_{\tau_1}(\text{OFF}) = 0 \leq 2$. Now consider the case $m = 2$. Since at time τ_1, ON has already bought a voucher v_0, $\Pi_{\tau_1}(\text{ON}) = \Pi(v_0)$. For OFF it either buys two vouchers v_0 with cost of $2\Pi(v_0)$ or buy a voucher v_j with cost $\Pi(v_0) < \Pi(v_j) \leq 2\Pi(v_0)$. Hence, we have $\Pi_{\tau_1}(\text{ON})/\Pi_{\tau_2}(\text{OFF}) \leq 2$.

Induction case ($m \geq 2$): According to the induction hypothesis, we assume that $\Pi_{\tau_{m-2}}(\text{ON})/\Pi_{\tau_{m-1}}(\text{OFF}) \leq 2$ and $\Pi_{\tau_{m-1}}(\text{ON})/\Pi_{\tau_m}(\text{OFF}) \leq 2$ and we intend to show $\Pi_{\tau_m}(\text{ON})/\Pi_{\tau_{m+1}}(\text{OFF}) \leq 2$. By our algorithm, $\Pi(v'_{m-1}) \leq \Pi_{\tau_{m-2}}(\text{ON})$ and $\Pi(v'_m) \leq \Pi_{\tau_{m-1}}(\text{ON})$. Note that no voucher satisfies $\Pi(v_i) \in (\Pi(v'_{m-1}), \Pi_{\tau_{m-2}}(\text{ON})]$ or $\Pi(v_i) \in (\Pi(v'_m), \Pi_{\tau_{m-1}}(\text{ON})]$ according to the algorithm structure.

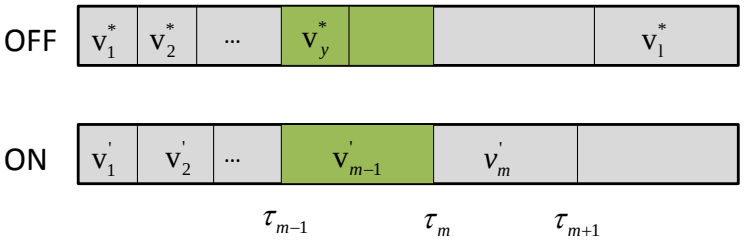
Now consider the vouchers that OFF buys to cover the interval $[\tau_1, \tau_{m+2})$. Assume that OFF purchases a sequence of vouchers $v_1^*,\ v_2^*,\ \cdots,\ v_l^*$. W.l.o.g., we assume $v_1^* \leq v_2^*,\ \cdots, \leq v_l^*$. We have $\Pi_{\tau_{m+1}}(\text{OFF}) = \Pi(v_1^*) + \cdots + \Pi(v_l^*)$. Now we consider the following three cases on the cost of the largest voucher v_l^* OFF buys:

1. $\Pi(v_l^*) \geq \Pi_{\tau_{m-1}}(\text{ON})$. We have $\Pi_{\tau_{m+1}}(\text{OFF}) \geq \Pi(v_l^*) \geq \Pi_{\tau_{m-1}}(\text{ON})$. Since $\Pi_{\tau_m}(\text{ON}) = \Pi_{\tau_{m-1}}(\text{ON}) + \Pi(v'_m) \leq 2\Pi_{\tau_{m-1}}(\text{ON})$, we have $\Pi_{\tau_m}(\text{ON})\ /\Pi_{\tau_{m+1}}(\text{OFF}) \leq 2$.

2. $\Pi_{\tau_{m-2}}(\text{ON}) \leq \Pi(v_l^*) < \Pi_{\tau_{m-1}}(\text{ON})$. In this case, recall that $\Pi(v_l^*) \notin (\Pi(v'_m), \Pi_{\tau_{m-1}}(\text{ON})]$ and $\Pi(v_l^*) \leq \Pi(v'_m)$. The example is shown in Figure 4.2a. Since $\Pi(v_l^*) \leq \Pi(v'_m)$, the requests with green are covered by vouchers $v_\alpha^*, \cdots, v_{l-1}^*$ where $1 \leq \alpha \leq l-1$. Therefore, $\Pi(v_1^*) + \cdots + \Pi(v_{l-1}^*) \geq \Pi_{\tau_m}(\text{OFF})$. Note that v_l^* is fully used. Moreover, by the induction hypothesis, $\Pi_{\tau_m}(\text{OFF}) \geq \Pi_{\tau_{m-1}}(\text{ON})/2$. Hence, we have $\Pi(v_1^*) + \cdots + \Pi(v_{l-1}^*) \geq \Pi_{\tau_m}(\text{OFF}) \geq \Pi_{\tau_{m-1}}(\text{ON})/2$.



(a) In the Case when $\Pi_{\tau_{m-2}}(\text{On}) \leq \Pi(v_l^*) < \Pi_{\tau_{m-1}}(\text{On})$, We Have $\Pi(v_1^*) + \cdots + \Pi(v_{l-1}^*) \geq \Pi_{\tau_m}(\text{Off})$.



(b) In the Case when $\Pi(v_l^*) < \Pi_{\tau_{m-2}}(\text{On})$, There Is Always a v_y^* Bought by Off Whose End Point Lies within the Voucher v'_{m-1} Bought by On.

Figure 4.2: Example to Show Proof for Lemma 5.

Now we can compute $\Pi_{\tau_{m+1}}(\text{Off})$ as following:

$$\Pi_{\tau_{m+1}}(\text{Off}) = \Pi(v_1^*) + \cdots + \Pi(v_l^*) \tag{4.1}$$

$$\geq \Pi_{\tau_{m-1}}(\text{On})/2 + \Pi(v_l^*) \tag{4.2}$$

$$\geq (\Pi_{\tau_{m-2}}(\text{On}) + \Pi(v'_{m-1}))/2 + \Pi_{\tau_{m-2}}(\text{On}) \tag{4.3}$$

$$\geq \Pi(v'_{m-1}) + \Pi_{\tau_{m-2}}(\text{On}) \tag{4.4}$$

Equality (4.1) holds by the definition and Inequality (4.2) is obtained by the induction hypothesis as discussed before. We substitute $\Pi_{\tau_{m-1}}(\text{On})$ with $\Pi_{\tau_{m-2}}(\text{On})$ $+\Pi(v'_{m-1})$ by algorithm structure and derive Inequality (4.3). In Inequali-

ty (4.4), according to the algorithm structure of On, we have $\Pi_{\tau_{m-2}}(\text{On}) \geq \Pi(v'_{m-1})$.

Note that $\Pi_{\tau_m}(\text{On}) = \Pi_{\tau_{m-1}}(\text{On}) + \Pi(v'_m) \leq 2\Pi_{\tau_{m-1}}(\text{On}) = 2(\Pi_{\tau_{m-2}}(\text{On}) + \Pi(v'_{m-1}))$. Hence, we have $\Pi_{\tau_m}(\text{On})/\Pi_{\tau_{m+1}}(\text{Off}) \leq 2$.

3. $\Pi(v_l^*) < \Pi_{\tau_{m-2}}(\text{On})$. Recall that $\Pi(v_l^*) \notin (\Pi(v'_{m-1}), \Pi_{\tau_{m-2}}(\text{On})]$, hence $\Pi(v_l^*) \leq \Pi(v'_{m-1})$ and $v_l^* \leq v'_{m-1}$. Note that v_l^* is the largest voucher among those Off buys. Now consider the interval $[\tau_m, \tau_{m+1}]$. Since no voucher $v_1^*,\ v_2^*,\ \cdots,\ v_l^*$ can cover the requests that start before τ_{m-1} and end after τ_m (otherwise it violates $\Pi(v_l^*) \leq \Pi(v'_{m-1})$), there exists a y such that the last time voucher v_y^* is used is within the interval $[\tau_{m-1}, \tau_m]$, where $1 \leq y < l$ showed in Figure 4.2b. By the induction hypothesis, we have $\Pi(v_1^*) + \cdots + \Pi(v_y^*) \geq \Pi_{\tau_{m-2}}(\text{On})/2$. On the other hand, the interval that the vouchers v_{y+1}^* to v_l^* cover is no less than the interval $[\tau_m, \tau_{m+1}]$, therefore, we have $\Pi(v_{y+1}^*) + \cdots + \Pi(v_l^*) \geq \Pi(v'_m)$ and $\Pi_{\tau_{m+1}}(\text{Off}) \geq \Pi_{\tau_{m-2}}(\text{On})/2 + \Pi(v'_m)$. Hence $\Pi_{\tau_m}(\text{On}) = \Pi_{\tau_{m-1}}(\text{On}) + \Pi(v'_m) = \Pi_{\tau_{m-2}}(\text{On}) + \Pi(v'_{m-1}) + \Pi(v'_m) \leq \Pi_{\tau_{m-2}}(\text{On}) + 2\Pi(v'_m)$.

Therefore, it holds for all the cases that $\Pi_{\tau_m}(\text{On})/\Pi_{\tau_{m+1}}(\text{Off}) \leq 2$. □

Theorem 8. *Algorithm* On *is 4-competitive.*

Proof. According to Lemma 5, within each discount window we have:

$$\begin{aligned}\rho &\leq \max_m \Pi_{\tau_m}(\text{On})/\Pi_{\tau_m}(\text{Off}) = (\Pi_{\tau_{m-1}}(\text{On}) + \Pi(v'_m))/\Pi_{\tau_m}(\text{Off}) \\ &\leq 2\Pi_{\tau_{m-1}}(\text{On})/\Pi_{\tau_m}(\text{Off}) \leq 4.\end{aligned}$$

We have already shown above that the competitive ratio within each discount window is bounded by 4. Since it is independent among all the discount windows, we can know that Algorithm On is 4-competitive. □

4.4.2 *Once-Expired-Lose-Discount (OELD)*

In contrast to the OELE model, the OELD model allows to keep the face value of the voucher, to buy goods at a price without discounts. In order to solve the OELD problem, we only slightly modify the algorithm On: our proposed algorithm OELD is essentially On, with the obvious improvement that new vouchers are only bought once remaining face values are used up.

Note that after a voucher expires, we lose discount for the remaining unused value. For example (Figure 4.3), a voucher v for 6 days yoga visit costs 60 dollars and it is valid before an expiration date, i.e., May 1^{st} 2015, while a single yoga visit costs 20 dollars. After May 1^{st} when the voucher expires, 4 visits have been used and the remaining value for the unused 2 visits becomes 20 dollars, which equals to 1 single visits, instead of 2 visits. That is, the remaining value is the real cost in proportional to the unused visits. As shown in Figure 4.3, when there is one future request coming in the second discount window, OELD can take advantage of the remaining face value to cover the new coming request while OELE has to buy a new single visit instead.

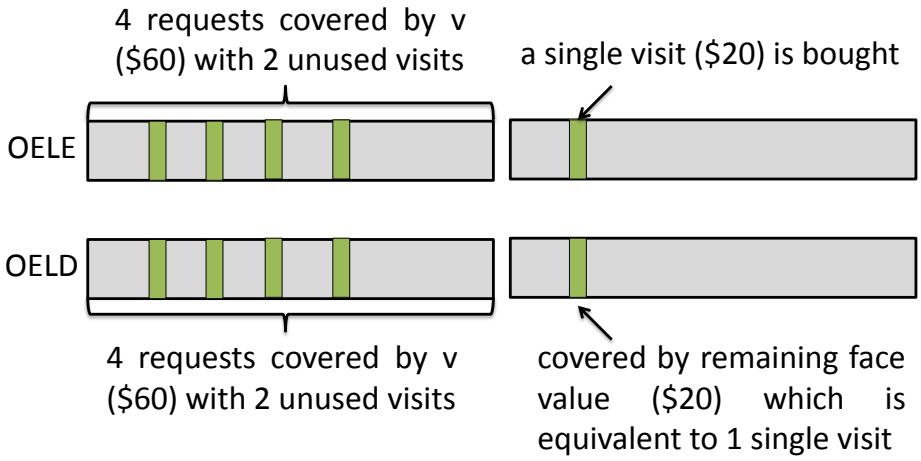


Figure 4.3: Example to Show that the Optimal Solution for OELD Is at least Half of That for OELE Given a Voucher $v = 6$ with $\Pi(v) = 60$ and $\Pi(v_0) = 20$.

Theorem 9. *Algorithm* On *is* 8*-competitive.*

Proof. We first prove the following helper lemma: it shows that by introducing remaining face value without discount in OELD, the optimal solution to OELE is not far away from that to OELD.

Lemma 6. *Let* Off_1 *be the optimal solution to our voucher purchasing problem under OELD model. And let* Off_2 *be the optimal solution for OELE. By introducing remaining face value without discount,* Off_1 *is a constant factor away from* Off_2*, i.e.,* $\Pi(\text{Off}_2) \leq max_i(2 - \frac{v_{i-1}}{v_i})\Pi(\text{Off}_1)$.

Proof. Let Off_1 be the optimal solution for OELD. W.l.o.g, Off_1 starts to use a voucher until it is fully used, then turns to buy next one. Thus all the other vouchers are fully consumed except for the last voucher within each discount window. Such optimal solution exists since if there is a voucher v in Off_1 which is fully unused within the current discount window, we can buy single visits instead with the same cost to cover those requests in future discount windows covered by the remaining value from v.

First let us construct an intermediate solution Off' as following: for each request covered by the remaining value in Off_1, replace it with a newly bought single visit. As discussed in our example above, the newly bought single visits add extra cost into Off', while the same amount of cost has already been paid by purchasing large voucher with some unused visits after the expiration date. We claim that for any voucher v_i in Off_1 with unused remaining value where $i > 1$, the used number of visits must be no less than v_{i-1}. Otherwise, Off_1 can buy v_{i-1} and single visits for the requests covered by v_i instead with less cost, which violates the optimality of Off_1. Therefore, the minimum used value is $\frac{v_{i-1}}{v_i}\Pi(v_i)$, and the maximum remaining value without discount can be computed by $(1 - \frac{v_{i-1}}{v_i}) \cdot \Pi(v_i)$. This guarantees that

for each such v_i, the cost OFF$'$ pays is the summation of the cost for v_i ($\Pi(v_i)$) and the cost for the newly bought single visits (at most $(1 - \frac{v_{i-1}}{v_i}) \cdot \Pi(v_i)$), that is OFF$'$ pays at most $(1 - \frac{v_{i-1}}{v_i}) \cdot \Pi(v_i) + \Pi(v_i) = (2 - \frac{v_{i-1}}{v_i}) \cdot \Pi(v_i)$. Note that $0 < \frac{v_{i-1}}{v_i} < 1$. Summing up the cost for all vouchers, we have $\Pi(\text{OFF}') \leq max_i(2 - \frac{v_{i-1}}{v_i})\Pi(\text{OFF}_1)$.

Now consider the optimal solution OFF_2 for OELE. It is obvious to see that in OFF$'$, the remaining value is not reused. Given the optimality of OFF, we can obtain that $\Pi(\text{OFF}_2) \leq \Pi(\text{OFF}')$. Therefore, we have $\Pi(\text{OFF}_2) \leq \Pi(\text{OFF}') \leq max_i(2 - \frac{v_{i-1}}{v_i})\Pi(\text{OFF}_1)$. □

Now consider applying the online algorithm OELD shown in Algorithm 8 to the OELD model. OELD can be seen as an online algorithm for OELD which does not use the remaining face value. Theorem 9 shows that OELD is an 8-competitive online algorithm for OELD.

According to Theorem 8, ON is 4-competitive for OELE. Moreover, Lemma 6 indicates that in the worst case the optimal solution for OELD costs at least half of that for OELE. Hence we have: $\rho = \Pi(\text{ON})/\Pi(\text{OFF}_1) \leq 2\Pi(\text{ON})/\Pi(\text{OFF}) \leq 8$. □

4.4.3 Limited Purchasing Window (LPW)

Note that the major difference between OELD and LPW lies on the length of the purchasing window. In OELD model, the purchasing window is same as the discount window, therefore, customers can buy vouchers anytime before the expiration date. However, in LPW model, the purchasing window is limited and customers need to decide to buy which vouchers and how many of them without knowing information about future requests. For any online algorithm, there must be a certain request sequence such that for any large vouchers bought within the purchasing window, there will be no more future requests to consume them within the same discount window. Therefore, for those vouchers, all the visits will lose discount and be treated

as single visits. That is, for an online algorithm which buys single visits for all the requests will give us a competitive ratio at the discount factor. Theorem 10 formally describes this observation.

Theorem 10. *The lower bound for any algorithm equals the discount factor.*

Proof. Assume that there is a smart adversary and the adversary follows two basic principles: 1. The adversary will never schedule demand within a purchasing window and 2. The adversary never schedules demand whenever the algorithm has a valid voucher for an discount window. If the algorithm buys a voucher within a discount window, the adversary will not schedule any demand, causing the voucher to be left unused within the period and therefore losing its discount factor. If the algorithm buys no voucher within the purchasing window, the adversary schedules demand for the rest of the discount window which can only be served by vouchers with no discount (either remaining or newly bought). OFF on the other hand will buy a voucher for this discount window to cover the demand after the purchasing window, which leads to a lower bound equal to the discount factor. □

4.5 Optimal Offline Algorithm

In this section, we present an efficient algorithm to solve the offline version of discount voucher purchase problem. We show that the solution could be computed by a dynamic programming algorithm in polynomial time. We prove the optimality of the algorithm for OELE and LPW separately and show the algorithm is a 2-approximation solution for OELD.

4.5.1 *Once-Expired-Lose-Everything (OELE)*

We present an optimal solution for this variant, shown in Algorithm 9, which is a dynamic programming based algorithm with a polynomial time complexity. In order to analyze the time complexity of the offline algorithm, we assume that there are n requests in σ, i.e., $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.

The basic idea behind the offline algorithm is that the optimal cost for any number of requests within the same discount window is obtained by the summation of the costs of two parts: one for buying a voucher to cover a certain amount of requests while the other one is the cost to cover the previous amount of request. Note that the optimal solutions to this smaller number of requests have already been computed. Since the purchasing behavior for each discount window is independent, we can compute the optimal solution for each discount window respectively.

First, let us discuss the data structure used in Algorithm 9. We recompute the request sequence and store the number of requests within each discount window in an array R. Assume that there are m such discount windows. Therefore, $R[s]$ represents the number of requests within the sth discount window, where $1 \leq s \leq m$. We store the optimal solution for the sth discount window in an array C_s of size $R[s] + 1$, in which the entry $C_s[i]$ indicates the optimal solution to the requests from the beginning to time σ_i, for $1 \leq i \leq n$. Initially, $C_s[0]$ is set to 0 and $C_s[i]$ is set to infinity for $i \in [1, R[s]]$. The summation of optimal solutions for all discount windows yields the optimal solution, which is denoted as C in the algorithm.

The algorithm proceeds as follows: It computes the optimal solution for the number of requests n' starting from 1. When the number of requests is x, where $x > 1$, the optimal solution can be computed by picking such a voucher g that the summation of the voucher cost $\Pi(g)$ and the cost for the uncovered requests by g (denoted as

$C_s[x-g]$ for certain s) is minimum. Note that $C_s[x-g]$ has already been computed optimally, this summation will yield the optimal solution for $n' = x$.

Algorithm 9 Offline Algorithm OFF

Input: Request sequence $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.

Output: Total Cost.

1: Let $R[1], \cdots, R[m]$ be the number of requests within each discount window and m be the number of discount windows
2: $C \leftarrow 0$
3: **for** $s = 1$ to m **do**
4: $\quad C_r[0] \leftarrow 0$
5: $\quad$ **for** $i = 1$ to $R[s]$ **do**
6: $\quad\quad C_s[i] \leftarrow \infty$
7: $\quad$ **for** $i = 1$ to $R[s]$ **do**
8: $\quad\quad$ **for** $j = 1$ to k **do**
9: $\quad\quad\quad l \leftarrow \max\{0, i - v_j\}$
10: $\quad\quad\quad C_s[i] \leftarrow \min\{C_s[i], C_s[l] + \Pi(v_j)\}$
11: $\quad C \leftarrow C + C_s[R[s]]$
12: **return** C

Theorem 11 shows the optimality and the time complexity of this algorithm.

Theorem 11. *Algorithm* OFF *computes an optimal offline solution to OELE in* $O(nk)$ *time.*

Proof. Correctness: Due to the independency among discount windows, we just need to prove the correctness for each discount window. Given a request sequence in the sth discount window, we prove the correctness of our algorithm by induction on i, the number of the requests. In the base case of $i = 1$, it is trivial to see that the

minimum cost to cover a request is the cost of the minimum voucher, i.e., $\Pi(v_0)$. In Line 9, l will be 0 since the length of any voucher is at least 1. Hence, since initially $C_r[1]$ is infinity, $C_s[0] + \Pi(v_0)(= \Pi(v_0))$ will give the optimal solution for the first request in Line 10.

Now we consider the inductive case when $i \geq 2$. The optimal solution is obtained by buying a voucher, say v_j, which minimizes the total cost of the newly bought voucher v_j and the optimal cost to cover the other $i - v_j$ requests. Since the new voucher will cover v_j requests, there are $l = \max\{i - v_j, 0\}$ requests uncovered by v_j. According to the induction hypothesis, $C[l]$ stores the optimal solution to cover $l = \max\{i - v_j, 0\}$ requests since $l \leq i$. Therefore, the minimum value of $C_s[l] + \Pi(v_j)$ over all j, where $1 \leq j \leq k$, computes the optimal cost of covering i requests. We then know that $C_s[R[s]]$ is an optimal solution to our problem in the current discount window. Hence, the summation of $C_s[R[s]]$ for all $s \in [1, m]$ will give the optimal solution (Line 11).

Time complexity: Now let us consider the time complexity of our algorithm. The precomputation in Line 1 takes $O(n)$ time. For the sth discount window, the initialization part of our algorithm (Lines 4-6) takes $O(R[s])$ times. The two for-loops in Lines 7 and 8 take $O(R[s]k)$ time. Since there will be m discount windows (Line 3), the total time is $n + \sum_{s=1}^{m} R[s]k = n + nk$. Therefore, the total time complexity is $O(nk)$. □

4.5.2 Once-Expired-Lose-Discount (OELD)

Note that in Theorem 6 we have already shown that the optimal solution to OELE is not far away from that to OELD, i.e., the optimal solution to OELE is at most twice of that to OELD. In Theorem 12, we show that optimal solution On to OELE shown in Algorithm 9 gives a 2-approximation solution to OELD.

Theorem 12. *Algorithm 9 gives a 2-approximation solution to OELD in time $O(nk)$.*

Proof. Theorem 11 has already shown the optimality of Algorithm 9 for OELE. According to Lemma 6, in the worst case the optimal solution to OELD is at least half of that to OELE. Therefore Algorithm Off gives a 2-approximation algorithm for our problem in time $O(nk)$. □

4.5.3 Limited Purchasing Window (LPW)

For the offline algorithm, the request sequence is given in advance hence the behavior of the offline algorithm is the same as that of OELE. Theorem 13 proves the optimality of Off for LPW.

Theorem 13. *Algorithm 9 computes an optimal solution to our problem in time $O(nk)$.*

Proof. Since all the requests are known in advance for the offline algorithm, the decision to buy vouchers is not limited by the purchasing window any more. This variant shares the same offline algorithm as OELE. Therefore, Algorithm 9 also computes an optimal solution to LPW in $O(nk)$ time. □

4.6 Conclusion

This work studied the Discount Voucher Purchasing Problem and three variants of this problem:

OELE: Vouchers lose their entire value after expiration;

OELD: Vouchers lose their discount value after expiration;

LPW: Vouchers can only be bought during a certain time window.

We have presented an online algorithm On that can compute a 4-competitive solution to OELE and an 8-competitive solution to OELD. We have also proven a lower bound on online competitive ratio for LPW. In addition, we have presented an offline algorithm that can compute an optimal solution to OELE and LPW and a 2-competitive solution to OELD.

We propose the following three interesting research topics:

1. Online algorithms: Currently, the remaining value in OELD is not used. One possible future topic is to design an online algorithm which takes use of the remaining value.

2. Computational complexity: We have presented a 2-competitive optimal algorithm for OELD. It would be useful to have theoretical computational complexity analysis for this variant.

3. In this work, we have not evaluated the performance of our algorithm in simulation. In order to have some knowledge of the performance, we propose to evaluate the following metrics which can show the properties of our algorithms:

    - Different discount functions: Different discount functions affect the purchasing pattern of the online algorithm. We propose to evaluate the competitive ratio under different discount functions and compare the contracts purchased along the proceeding to the largest one for the online algorithm.
    - Different voucher sets: The behavior (e.g., competitive ratios) of the online algorithm under different voucher sets is an interesting research topic to explore.

Chapter 5

CONCLUSION AND FUTURE WORK

In this thesis, we have studied three critical problems in resource allocation.

In the first problem, we have discussed a scenario where a service can be seamlessly migrated closer to users for better quality of service. We have studied the strategies of when and where to migrate the service, and also how much resource to purchase while migrating the service. We have presented optimal migration contracts purchasing strategies under two different pricing models: Pay-as-You-Come and Pay-as-You-Go. We have also discussed two online algorithms and validated their quantified competitive results via simulation.

For future work, we propose the following two possible research directions for the first work:

1. Online algorithms: One interesting research task is to discuss the competitive online algorithm for our first problem.

2. Experiments: Currently, we propose two online algorithms and evaluate their competitive ratios with experiments. However, the settings used in the evaluation have some constraints. Therefore, another possible future work could be to investigate the competitive ratio under different simulation settings, i.e., different request models.

In the second problem, we have studied a cost-effective cloud resource allocation problem and introduced the 2-dimensional Parking Permit Problem PPP^2. We have showed that our problem can be seen as equivalent to the PPP^2 problem and we have presented an online algorithm with competitive analysis on its upper bound and

lower bound. We have also proposed an offline algorithm which the online algorithm relies on as a black box. At the end of this work, we have extended the 2-dimension problem to higher dimensions, introducing the D-dimension PPP problem, and have proposed an optimal algorithm.

We propose the following two interesting research topics:

1. Randomized algorithms: It would be interesting to investigate whether the randomized algorithms known for the classic parking permit problem can also be generalized to two or even higher dimensions.

2. Experiments: We have already evaluated the performance of our algorithms under the request model in which the rate the of the requests follows a uniform distribution. An interesting research direction is to evaluate the performance for other different distributions (e.g., normal distribution) and analyze how these request models affect the competitive ratios. Since different discount functions may affect the competitive results, it would be useful to simulate under different discount functions, such as logarithmic or square discount.

In the third problem, we have introduced the Discount Voucher Purchasing Problem and three variants of this problem:

OELE: Vouchers lose their entire value after expiration;

OELD: Vouchers lose their discount value after expiration;

LPW: Vouchers can only be bought during a certain time window.

We have presented an online algorithm On that can compute a 4-competitive solution to OELE and an 8-competitive solution to OELD. We have also proven a lower bound

on online competitive ratio for LPW. In addition, we have presented an offline algorithm that can compute an optimal solution to OELE and LPW and a 2-competitive solution to OELD.

We propose the following three interesting research topics:

1. Online algorithms: Currently, the remaining value (the value after expiration date and without discount) in OELD is not used. One possible future topic is to design an online algorithm which takes use of the remaining value.

2. Computational complexity: We have presented a 2-competitive optimal algorithm for OELD. However, we have not discussed the hardness to find an optimal solution. It would be useful to have theoretical computational complexity analysis for this variant.

3. In this work, we have not evaluated the performance of our algorithm in simulation. In order to have some knowledge of the performance, we propose to evaluate the following metrics which can show the properties of our algorithms:

    - Different discount functions: Different discount functions affect the purchasing pattern of the online algorithm. We propose to evaluate the competitive ratio under different discount functions and compare the contracts purchased along the proceeding to the largest one for the online algorithm.
    - Different voucher sets: The behavior (e.g., competitive ratios) of the online algorithm under different voucher sets is an interesting research topic to explore.

REFERENCES

[1] V. Abhishek, I. A. Kash, and P. Key. Fixed and market pricing for cloud services. In *Proc. NetEcon Workshop*, 2012.

[2] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: automated data placement for geo-distributed cloud services. In *Pro. 7th USENIX NSDI*, 2010.

[3] K. Ali, K. Murali, and P. N. P. Krishna. The constrained ski-rental problem and its application to online cloud cost optimization. In *Proceedings IEEE INFOCOM*, 2013.

[4] M. Armbrust, A. Fox, R. Grifith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. In *UC Berkeley Technical Report EECS-2009-28*, 2009.

[5] D. Arora, M. Bienkowski, A. Feldmann, G. Schaffrath, and S. Schmid. Online strategies for intra and inter provider service migration in virtual networks. In *Proc. IPTComm*, 2011.

[6] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proc. ACM SIGCOMM*, pages 242–253, 2011.

[7] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer. Minimum congestion mapping in a cloud. In *Proc. ACM PODC*, pages 267–276, 2011.

[8] A. Belbekkouche, M. Hasan, and A. Karmouch. Resource discovery and allocation in network virtualization. *IEEE Communications Surveys Tutorials*, (99):1–15, 2012.

[9] M. Bienkowski, A. Feldmann, J. Grassler, G. Schaffrath, and S. Schmid. The wide-area virtual service migration problem: A competitive analysis approach. *IEEE/ACM Transactions on Networking (ToN)*, 2013.

[10] M. Bienkowski, A. Feldmann, D. Jurca, W. Kellerer, G. Schaffrath, S. Schmid, and J. Widmer. Competitive analysis for service migration in VNets. In *Proc. ACM VISA*, pages 17–24, 2010.

[11] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[12] 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. *Elsevier FGCS*, 25(6), 2009.

[13] K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. IEEE INFOCOM*, 2009.

[14] N. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54:862–876, 2010.

[15] X. Chunlin, M. Weimin, and Y. Lei. Competitive analysis of two special online device replacement problems. *Journal of Computer Science and Technology*, 23(2):203–213, 2008.

[16] CloudNet. Migration Demo. In *http://www.youtube.com/watch?v=llJce0F1zHQ*, 2012.

[17] D. Dash, V. Kantere, and A. Ailamaki. An economic model for self-tuned cloud caching. In *Proc. IEEE International Conference on Data Engineering*, pages 1687–1693, 2009.

[18] G. Even, M. Medina, G. Schaffrath, and S. Schmid. Competitive and deterministic embeddings of virtual networks. In *Proc. 13th ICDCN*, 2012.

[19] R. Fleischer. On the bahncard problem. *Theor. Comput. Sci.*, 268(1):161–174, 2001.

[20] S. Goyal and B. Giri. Recent trends in modeling of deteriorating inventory. *Elsevier EJOR*, 134(1), 2001.

[21] S. Guha and K. Munagala. Improved algorithms for the data placement problem. In *Proc.13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 106–107, 2002.

[22] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *Proc. 6th CoNEXT*, 2010.

[23] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. R. K. Sripanidkulchai, and M. Tawarmalani. Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud. In *Proc. ACM SIGCOMM*, 2011.

[24] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song. Enhancing dynamic cloud-based services using network virtualization. *SIGCOMM Comput. Commun. Rev.*, 40(1):67–74, 2010.

[25] T. A. Henzinger, A. V. Singh, V. Singh, T. Wies, and D. Zufferey. A marketplace for cloud resources. In *Proc. 10th ACM EMSOFT*, 2010.

[26] F. Hiroshi, K. Takuma, and T. Fujito. On the best possible competitive ratio for multislope ski rental. *Algorithms and Computation*, 2011.

[27] R. M. Karp. Online algorithms versus off-line algorithms: How much is it worth to know the future? *IFIP 12th World Computer Congress*, pages 416–429, 1992.

[28] A. Kumar, A. Gupta, and T. Roughgarden. A constant-factor approximation algorithm for the multicommodity. In *Proc. 43rd Symposium on Foundations of Computer Science (FOCS)*, 2002.

[29] A. Ludwig, S. Schmid, and A. Feldmann. The price of specificity in the age of network virtualization (short paper). In *Proc. 5th IEEE/ACM UCC*, 2012.

[30] M. Armbrust et al. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[31] A. M. Meyerson. The parking permit problem. In *Proc. 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 274–284, 2005.

[32] J. C. Mogul and L. Popa. What we talk about when we talk about cloud network performance. *SIGCOMM Comput. Commun. Rev. (CCR)*, Sept. 2012.

[33] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proc. 5th ACM EuroSys*, pages 237–250, 2010.

[34] R. Pal and P. Hui. Economic models for cloud service markets. In *Proc. ICDCN*, 2012.

[35] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Buy-at-bulk network design: Approximating the single-sink edge installation problem. In *Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 619–628, 1997.

[36] I. Sandy and R. Dinesh. The problem of renting versus buying. In *personal communication*, 1998.

[37] G. Schaffrath, S. Schmid, and A. Feldmann. Optimizing long-lived cloudnets with migrations. In *Proc. 5th IEEE/ACM UCC*, 2012.

[38] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy. Network virtualization architecture: proposal and initial prototype. In *Proc. ACM VISA*, 2009.

[39] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy. Network virtualization architecture: Proposal and initial prototype. In *Proc. VISA*, 2009.

[40] M. Schönherr. T-Labs Berlin. In *Private Communication*, 2012.

[41] K. Sebastian, M. Christine, M. H. Friedhelm, and S. Christian. Algorithmic aspects of resource management in the cloud. *Structural Information and Communication Complexity*, 2014.

[42] S. Shakkottai and R. Srikant. Economics of network pricing with multiple isps. *IEEE/ACM TON*, 14(6), 2006.

[43] wiki. Groupon. In *http://en.wikipedia.org/wiki/Groupon*, 2015.

[44] wiki. Voucher. In *http://en.wikipedia.org/wiki/Voucher*, 2015.

[45] T. Wood, P. Shenoy, K. Ramakrishnan, and J. V. der Merwe. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proc. ACM VEE*, 2011.

[46] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: incorporating time-varying network reservations in data centers. In *Proc. SIGCOMM*, pages 199–210, 2012.

[47] G. Zhang, C. K. Poon, and Y. Xu. The ski-rental problem with multiple discount options. *Inf. Process. Lett.*, 111(18):903–906, 2011.

[48] S. Zhang, Z. Qian, J. Wu, and S. Lu. An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In *Proc. IEEE INFOCOM*, 2012.