2018-04-20

Efficient Algorithms for Dynamic Cloud Resource Provisioning

Zhou, Ruiting

UNIVERSITY OF CALGARY

Efficient Algorithms for Dynamic Cloud Resource Provisioning

by

Ruiting Zhou

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

April, 2018

Abstract

Cloud computing has emerged as a new computing paradigm, with data centers proliferating in today's Internet. Cloud service providers often adopt static resource provisioning to pack cloud resources to fixed types of virtual machines (VM), failing to address user demands efficiently and precisely. In this thesis, we focus on dynamic cloud resource provisioning, which provides real-time, on-demand access to cloud resources. We propose efficient algorithms to guide resource allocation and workload dispatching in cloud systems.

We first study dynamic VM provisioning via an online auction algorithm. We generalize the existing literature by introducing computing jobs with completion deadlines. A cloud user bids for future cloud resources to execute its job. Each bid specifies (a) a resource profile of tailor-made VMs, (b) a utility, reflecting the amount that the user is willing to pay for executing its job, and (c) a soft deadline, specifying the preferred finish time of the job, as well as a penalty function that characterizes the cost of violating the deadline. We propose efficient cloud job auctions that execute in an online fashion, provide truthfulness guarantee, and achieve a good competitive ratio.

We then discuss cloud container services, a more recent form of cloud resource provisioning. Compared to traditional VMs, cloud containers are more flexible and lightweight. We exploit this new algorithm design space, and study dynamic cloud container provisioning. We design efficient scheduling algorithms for complex computing jobs that are running on cloud containers. Our offline and online schedulers permit partial execution, allow a job to specify its job deadline, desired cloud containers, and inter-container dependence relations, and achieve near-optimal expected objective values.

We further extend our study to cloud container clusters. Enterprise users often create clusters of inter-connected containers to provision complex services. Compared to traditional cloud services, key challenges in container cluster (CC) provisioning lie in the optimal placement of containers while considering inter-container traffic in a CC. The challenge further escalates when CCs are

provisioned in an online fashion upon CC request arrivals. We investigate dynamic cloud CC provisioning, and propose an online algorithm to address the above challenges. Our online algorithm achieves computational and economical efficiencies.

Acknowledgments

First and foremost, I would like to sincerely thank my supervisor, Dr. Zongpeng Li. I cannot express my appreciation and thanks enough, for his invaluable guidance, advice and support during my PhD study at the University of Calgary. Zongpeng sparked my interest in the area of computer networks and helped me become a mature researcher. I was so grateful for his innovative ideas and consistent feedback throughout my PhD study. Moreover, Zongpeng has been not only my teacher, but also my mentor and advisor in every aspects of my life. Working together with Zongpeng has been the best experience that I will cherish for ever.

I would also like to thank Dr. Chuan Wu for her continual support and guidance. She took part in all my projectors and always gave me insightful comments. I am truly fortunate to have the opportunity to work with her. I would like to thank my PhD supervisory committee members, Carey Williamson, Philipp Woelfel, Diwakar Krishnamurthy, and Kui Wu for their time and helpful comments. I am also grateful for my friends and colleagues in the Department of Computer Science. They gave me a good time in Calgary. Last but not the least, I would like to express my heartfelt thanks to my family for their endless love and unconditional support.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
VM	Virtual Machine
CC	Container Cluster
NFV	Network Function Virtualization
LP	Linear Program
ILP	Integer Linear Program
IP	Integer Program
IQP	Integer Quadratic Program
VC	Virtual Cluster
IDS	Intrusion Detection System

Chapter 1

Overview

Resource allocation problems in cloud computing systems have attracted substantial attention in the recent literature. This thesis focuses on the design of efficient algorithms to dynamically allocate cloud resources. In this chapter, we introduce the background of cloud computing, describe the problem of dynamic cloud resource provisioning, and overview the structure of this thesis.

Cloud computing has emerged as a new computing paradigm that offers users rapid on-demand access to shared pools of configurable resources such as CPU, RAM and disk storage, with minimal management effort. According to a recent report, global cloud service providers account for approximately $260 billion of revenue, from cloud computing services by the end of 2017, up from $219.6 billion in 2016, and will reach $411 billion by 2020 [34]. Meanwhile, the energy consumption of data centers rises rapidly. Data centers consumed 1.5% of all electricity worldwide in 2011, and the ratio is predicted to increase to 8% by 2020 [50]. A small fraction of improvement in system efficiency would lead to considerable revenue improvement and energy saving.

In the past decade, two types of cloud platforms blossomed on the Internet, including (i) large-scale Internet data centers, exemplified by Amazon EC2 [1], Microsoft Azure and Linode [5, 6], which organize a shared resource pool for serving their users; and (ii) co-location data centers, often found in metropolitan areas, where smaller clouds from different users are physically co-located, jointly managed and serviced by the co-location [79]. Our proposed algorithms can be applied to both platforms.

Cloud service providers often pack their computing resources into different types of virtual machine (VM), to serve different cloud jobs. For example, Amazon EC2 [1] offers 18 VM instance types in 5 categories. Table 1.1 illustrates selected VM types available at Amazon EC2. Each type of VM has its focus and forte, and a large computing job often requires cooperation among multiple

VM instances. For example, social games [60] and enterprise applications [41] are often composed of a front-end web server tier, a load balancing tier and a back-end data storage tier, each suited for execution on a VM that is abundant in a particular type of resource: bandwidth, CPU, and storage, respectively.

Table 1.1: Amazon EC2 virtual machines instance types

VM type	vCPU	Memory(GiB)	Storage (GB)	Networking Performance
m3.medium	1	3.75	1×4 SSD	Moderate
m3.2xlarge	8	30	2×80 SSD	High
c3.large	2	3.75	2×6 SSD	Moderate
c3.4xlarge	16	30	2×160 SSD	High
r3.4xlarge	16	122	1×320 SSD	High
d2.2xlarge	8	61	6×2000 HDD	High

Currently, cloud service providers often adopt static resource provisioning, where VM instances are provisioned in advance with a fixed number of VMs for each type. Although cloud service providers have been expanding the variety of VM instances, sometimes they still fail to meet the exact needs, leading to a waste of resources and an unbalanced matching between user demands and available resources. For example, assume a user needs to deploy a MapReduce service in Amazon's data centers. Its job requires 8 vCPU and 16 GB memory to process 80 GB usage data [14]. Among all VM instance types Amazon EC2 can provide, m3.2xlarge is the best match. However, it is still far away from a perfect match and roughly half of the allocated memory and SSD storage are wasted. Recently, some cloud platforms start to offer customized VMs that pack various resources at user-specified amounts [2] [7]. Under dynamic provisioning, cloud service providers assemble customized VMs in an online fashion upon requests, capturing the dynamic fluctuation of user demands.

In this thesis, we focus on the efficient algorithm design for dynamic cloud resource provisioning. We aim to design general and expressive algorithms that can help cloud service providers more efficiently allocate their resources, to improve system stability and achieve maximum possi-

ble social welfare. We not only study the dynamic provisioning of VMs, but also pay attention to the newly emerged cloud container services. Compared to VMs, cloud containers do not require a full, dedicated operating system to be installed within them, and can launch in microseconds [76]. Cloud containers are more flexible and lightweight, offering an effective alternative for cloud resource provisioning. The following three problems are investigated in this thesis: i) Online auction algorithm design for cloud computing jobs with deadlines; ii) Scheduling framework design for cloud container services; iii) Online placement scheme design for cloud container clusters.

In what follows, we provide a brief introduction to each of the problems described, and list our contributions.

1.1 Online Auction for Cloud Computing Jobs with Deadlines

Cloud computing jobs can be categorized into two types, depending on whether their computing need is elastic or not. Cloud jobs such as large scale web servers utilize cloud service as a utility, and require the rented VMs to be always active, with possible dynamic size scaling. These jobs are similar to the *power users* in a power grid, who demand always-on power supply. Other jobs such as big data analytics and Google crawling data processing often have a batch processing nature. They require a certain computing job to be completed without demanding always-on VM service, and may tolerate a certain level of delay in job completion. These users are similar to the *energy users* in a power grid who need to draw a fixed quantity of energy for powering a given job, but in a flexible time window.

Today's cloud providers often adopt a fixed price policy and charge users a fixed amount per-VM usage. Despite their apparent simplicity, fixed-price policies inherently lack market agility and efficiency, failing to rapidly adapt to real-time demand-supply fluctuations. Consequently, overpricing and underpricing routinely occur, which either dispel or undercharge the users, jeopardizing overall system social welfare as well as the providers' revenue. In contrast, a well designed auction mechanism can automatically discover the right market price and dynamically allocate

resources according to user demands.

Existing market mechanisms for cloud computing, particularly the auction type mechanisms, have been implicitly targeting the case of non-elastic cloud jobs. In such one-round [78] and online [62] cloud resource auctions, once a bid is accepted, the service time window of the corresponding VMs is fixed, *i.e.* either in the current round [62] or between the start and finish times prescribed in the bid [80]. Such auction algorithms do not need to consider the scheduling of accepted jobs. In contrast, a well designed market mechanism for elastic jobs must pay close attention to not only whether to accept a bid, but when to schedule its execution based on its deadline information. For example, consider a cloud user who bids for a VM bundle tailored for human genome analysis. Its job can be processed within 3 hours if the specified VM bundle is provisioned; however, as long as the computing result is available within the next 24 hours, the user is happy. This leaves ample space for job scheduling in the temporal domain, which a well-designed auction algorithm should judiciously exploit to maximize resource utilization and social efficiency — for example, scheduling a job within its tolerance window to time slots with relatively low demand. In Chapter 3 of the thesis, we focus on elastic computing jobs that have completion deadlines, and propose an online auction to address the dynamic provisioning of VMs. More specifically, our contribution can be stated as follows:

- We propose a new framework, *compact-exponential* linear programs (LPs), which works in concert with a dual oracle to handle non-conventional deadline constraints.
- We first consider a basic setting, where each job has several alternative hard deadlines. By combining the compact-exponential technique with the classic primal-dual method, we are able to adapt the recent posted-pricing auction framework to design an efficient online cloud auction that guarantees truthful bidding, and achieves near-optimal social welfare.
- We further generalize our cloud auction design by considering server cost and the general form of a soft deadline. We propose a truthful online auction that achieves

computational efficiency and a good competitive ratio.

- We conduct simulation studies based on real world trace data. We show that our online auctions perform better than the theoretical bound, and can produce between $25\% - 80\%$ of the optimal social welfare.

1.2 Scheduling Frameworks for Cloud Container Services

Cloud resources, including CPU, RAM, disk storage and bandwidth, used to be packed into different types of VMs to serve different computing jobs. Launching a VM instance requires running a full, dedicated operating system, which often consumes extra resources and takes minutes or even longer [73]. More recently, cloud containers offer an alternative to VMs, providing a streamlined, easy-to-deploy method of resource management. Relying on encapsulated applications, container service requires no dedicated operating system. A cloud container is flexible, operates with the minimum amount of resources and starts in microseconds [76]. Container services available on the cloud market today include Google Container Engine [36], Amazon EC2 Container service (ECS) [11], Aliyun Container Service [10], and Azure Container Service [54].

A complex computing job consists of multiple subtasks, each requiring a different configuration of cloud resources. A customized cloud container can be created accordingly to serve each subtask based on a user-defined resource profile. A subtask may depend on another, and can start execution only after the latter is completed. Such dependencies can be captured by a *dependence* graph. For example, a service chain in Network Function Virtualization (NFV) is composed of a sequential chain of virtualized network functions (VNFs) [39]. An image rendering job creates a 2D raster representation of a 3D model. It is composed of four subtasks to be executed sequentially: vertex processing, clipping and primitively assembling, rasterizing and fragment processing [70]. Tailor-made cloud services are available to such jobs. For instance, Azure Batch [56] is a service from Microsoft Azure, for batch processing in the cloud. A user first creates a batch job in its account and then initializes the job, including creating subtasks, configuring the container for each subtask,

defining schedules and dependencies of subtasks.

While some computing jobs are time-sensitive, requiring full execution before the deadline, other jobs are elastic, and can be partially executed to obtain partial values. For example, a partially completed web searching job may return the top search results in a short time period, which is often good enough for the users [81]. After finishing the first subtask in an image rendering job, the shape of the 3D model has been outlined by vertices [70], which already provides useful information to the user. The new model of partial value for partial execution is first described as a Quality-of-Service (QoS) problem concerning the visualization of large images across a network [25]. It has applications in numerical computation, heuristic search, and database query processing [26]. Scheduling of computing jobs with partial values in the cloud has attracted recent attention from the literature [18, 53, 81, 82].

In Chapter 4 of the thesis, we study cloud container services, and propose scheduling algorithms that allow partial execution, and process complex computing jobs. Our contribution can be summarized as follows:

- We extend the existing literature on dynamic cloud resource provisioning, and present the first study that investigates the scheduling design for cloud container services.

- Our schedulers are expressive enough such that they permit partial execution and can handle general type of jobs, *i.e.,* jobs with interrelated subtasks.

- We exploit the compact-exponential technique, and combine it with randomized rounding and online learning techniques, to design scheduling algorithms. We prove that both the offline and the online scheduling frameworks guarantee computational efficiency, and produce near-optimal expected objective values.

- Trace-driven simulation studies verify that our scheduling algorithms can achieve close-to-optimal objective value in short computation time.

1.3 Online Placement Scheme for Cloud Container Clusters

Besides purchasing individual containers, cloud users often require a collection of containers and the network in between, to create a container cluster (CC) for their jobs or services. Typical examples include MapReduce programs that process big data with distributed algorithms running on containers in a CC, and service chains in a Network Function Virtualization (NFV) environment. A service chain refers to the structure of a network service where a sequence of virtual network functions are linked [39]. For example, an enterprise may request a CC to deploy an access service chain "Firewall→IDS→Proxy", where instances of firewall, intrusion detection system (IDS) and proxy are encapsulated into containers. Container clusters are emerging as the new form of virtual clusters. Compared to traditional virtual clusters, container clusters, *e.g.,* Google Container Cluster [35], Amazon ECS Cluster [12] and Azure Container Service Cluster [55], provide better performance for applications and enhance the elasticity by fast deployment of additional work nodes.

In Chapter 5 of this thesis, we target a more realistic and general setup in the deployment of CCs. We investigate the online CC placement problem that dynamically assembles CC as per user request. We take the perspective of a cloud service provider, who hosts cloud computing resources in multiple *zones*, where a zone may correspond to one or multiple servers, or a data center. The computing resources in a region owned by Amazon, for instance, are divided to *Availability Zones* [13]. The cloud service provider deploys containers and assembles CC upon requests on the fly. The deployment of a CC involves not only the placement of containers, *i.e.,* assigning each container to a zone with free capacity, but also routing inter-container traffic, *i.e.,* identifying zones with available bandwidth in between to send traffic between neighbor containers. Even in the offline setting with full information, such a deployment problem translates into an NP-hard combinational optimization problem. The challenge further escalates when we target a practical online placement scheme that makes on-spot decisions upon the arrival of each CC request. We propose an efficient online placement scheme to address the above challenges. In particular, our

contributions are listed as follows:

- We jointly optimize the placement of containers and the routing of inter-container traffic while placing CCs.
- A one-shot CC placement problem that determines a given CC's placement scheme is studied. We propose an efficient approximation algorithm to solve this problem, with the goal of cost minimization.
- We then apply compact-exponential and primal-dual techniques to design an online placement scheme that employs the one-shot algorithm as a building block to make decisions upon the arrival of each CC request.
- Through both theoretical analysis and trace-driven simulations, we verify that our online placement algorithm is computationally efficient and achieves a good competitive ratio.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 introduces preliminaries in auction design and algorithm design. Before we present a cloud auction in Chapter 3, we first provide an introduction to some of the fundamental concepts in auction theory. We then introduce some useful definitions in optimization algorithm design.

Chapters 3-5 present our main technical results, corresponding to each of the problems described in Ch. 1.1-Ch. 1.3. In Chapter 3, we study the cloud market for computing jobs with completion deadlines, and design efficient online auctions for VM provisioning. This is joint work with Zongpeng Li, Chuan Wu and Zhiyi Huang, and is published in IEEE/ACM Transactions on Networking (ToN) [84]. Chapter 4 proposes both offline and online scheduling frameworks for cloud container provisioning. The work presented in Chapter 4 is completed in collaboration with Zongpeng Li and Chuan Wu, and appears in IEEE/ACM ToN [83]. Chapter 5 studies the online

algorithm design for placements of cloud container clusters. The results are done in collaboration with Zongpeng Li and Chuan Wu, and were submitted to IEEE Journal on Selected Areas in Communications (JSAC) in April 2018.

Chapter 6 concludes this thesis by summarizing the results and discussing directions for further research.

Chapter 2

Preliminaries

We first introduce some fundamental concepts in auction theory in Ch. 2.1, and then describe some useful definitions related to algorithm design in Ch. 2.2.

2.1 Preliminaries in Auction Design

We begin with a general auction. An auction is a process where the bidders (buyers) buy goods or resources from an auctioneer (seller) by offering prices. The auctioneer then makes two decisions - which bidders win, and how much they pay for the goods or resources. Next, we will introduce some concepts in the auction theory. Every bidder has a valuation for the goods or resources being auctioned, in other words, the maximum amount they are willing to pay to get them. A bidder's utility equals its valuation minus its payment (how much it is charged) if it wins, and zero otherwise. Every bidder aims to maximize its own utility. In summary, we have the following terms:

- n: the number of bidders.
- b_i: the bidding price of the i-th player.
- v_i: the valuation of the i-th player.
- p_i: the price that the i-th player is charged.
- x_i: the outcome of the auction; $x_i = 1$ if the i-th player wins and $x_i = 0$ otherwise.
- u_i: the utility of the i-th player; $u_i = (v_i - p_i)x_i$.

The mechanism of an auction consists of the following elements:

- The bidding procedure.
- A, the allocation function, which determines who receives the goods or resources, *i.e.,* the value of x_i.
- P, the price function, which determines the payment, *i.e.,* value of p_i.

In terms of strategic behaviour, bidders are assumed to be rational but selfish, with a natural goal of maximizing their respective utilities. They may choose to submit a falsified bid $b_i \neq v_i$, if doing so may lead to a higher utility. We instead value the "happiness" of the entire ecosystem, and pursue highest social welfare possible, for which it is important to elicit truthful bids from users.

Definition 1. (Dominant Strategy [37]): A strategy is dominant if the strategy earns a user a larger utility than any other strategies, regardless of what any other users do.

Definition 2. (Truthful Auction [37]): An auction is truthful if for any bidder i, bidding its true valuation v_i forms its dominant strategy, *i.e.,* for all $b_i \neq v_i$, $u_i(v_i) \geq u_i(b_i)$.

A social welfare maximization mechanism aims to maximize the total utility of all bidders, as well as the utility of the auctioneer. The auctioneer's utility is the sum of the payments received, *i.e.,* $\sum_{i=1}^{n} p_i$. The social welfare is defined as follows:

Definition 3. (Social Welfare [37]): The social welfare is the aggregate utility of the auctioneer ($\sum_{i=1}^{n} p_i$) and bidders ($\sum_{i=1}^{n}(v_i x_i - p_i)$). With payments cancel themselves, the social welfare is simply $\sum_{i=1}^{n} v_i$.

In the next chapter, we design a cloud auction that aims to maximize the social welfare and guarantee truthful bidding. We will specify the detailed definitions of truthfulness and social welfare in a cloud auction in Ch. 3.3. Note that maximizing social welfare and maximizing seller profit are both natural, conceivable and interesting research problems in essentially all real world auction design. When social welfare is maximized, the overall eco-system is most efficient and there is a higher probability for everyone to be happy in the long run. In a cloud system, with the presence

of multiple cloud service providers, if a particular cloud provider wishes to keep its cloud users, it may want to pay attention to its users' happiness besides caring about its immediate profit.

2.2 Other Definitions in Algorithm Design

In this thesis, we formulate problems to integer linear programs, including in particular 0-1 integer linear programs of the packing type. The formal definition is [72]:

Definition 4. A linear program is a packing 0-1 integer linear program if it has the following form: Maximize $\mathbf{b}^T\mathbf{x}$, subject to: $\mathbf{A}\mathbf{x} \leq \mathbf{c}$ and $\mathbf{x} \in \{0,1\}$, where $\mathbf{x}$ is the vector of variables, $\mathbf{b}$ and $\mathbf{c}$ are vectors of non-negative coefficients, $\mathbf{A}$ is a matrix of non-negative coefficients, and $(\cdot)^T$ is the matrix transpose. The expression to be maximized, $\mathbf{b}^T\mathbf{x}$, is called the objective function, and inequalities $\mathbf{A}\mathbf{x} \leq \mathbf{c}$ and $\mathbf{x} \in \{0,1\}$ are constraints that specify a convex polytope over which the objective function is to be optimized.

In computer science, approximation algorithms are efficient algorithms that generate approximate solutions to NP-hard optimization problems. When we analyze the performance of an approximation algorithm, the approximation ratio is used to measure the distance of the returned solution from the optimal one. Next, we introduce the following definitions for maximization problems and minimization problems, respectively [72].

Definition 5. An algorithm A for a maximization problem is called an α-approximation algorithm, if $\alpha A(I) \geq OPT(I), \forall I$, where $\alpha \geq 1$, $A(I)$ is the objective value returned by A to the problem under consideration for input I and $OPT(I)$ is the objective value of an optimal solution under input I.

Definition 6. An algorithm A for a minimization problem is called an α-approximation algorithm, if $A(I) \leq \alpha OPT(I), \forall I$, where $\alpha \geq 1$, $A(I)$ is the objective value returned by A to the problem under consideration for input I and $OPT(I)$ is the objective value of an optimal solution under input I.

Here, α is the approximation ratio. For a maximization problem, α is the upper bound ratio of the optimal objective value to the objective value achieved by algorithm A. For a minimization

problem, α is the upper bound ratio of the objective value achieved by algorithm A to the optimal objective value. In the design of approximation algorithms, we want to keep α as small as possible.

An online algorithm is an algorithm that can process a sequence of inputs or data arriving over time, and it needs to make decisions on the fly immediately without being able to see the future. In contrast, an offline algorithm can access the whole problem data from the beginning and is required to solve the problem at hand. Competitive analysis is a method for analyzing online algorithms, and the competitive ratio of an online algorithm is the ratio between its performance and the optimal offline algorithm's performance. Next, we define the concept of "competitiveness" for maximization problems as follows:

Definition 7. An online algorithm B for a maximization problem is β-competitive if $\beta B(I) \geq OPT_{offline}(I)$, where $\beta \geq 1$, I is a sequence of input, $B(I)$ is the objective value of the solution found by B and $OPT_{offline}(I)$ is the objective value returned by an optimal offline algorithm.

Here, β is the competitive ratio. For an online maximization problem, β is the upper bound ratio of the optimal objective value to the objective value returned by algorithm B, and a smaller β indicates better performance guarantee of the online algorithm.

Chapter 3

An Efficient Cloud Market Mechanism for Computing Jobs with Soft Deadlines

3.1 Introduction

In this chapter, we generalize existing auction design for dynamic cloud resource provisioning by proposing online auctions that explicitly handle jobs with prescribed deadlines. We further allow a cloud user to express soft deadlines, described by a preferred job completion time, coupled with a penalty function that encodes how much penalty is associated with different degrees of deadline violation. Compared with simple market mechanisms such as fixed pricing, a well-designed auction provides automatic price discovery, promptly adapts prices with the fluctuation of supply and demand, and allocates cloud resources to jobs who value them the most, maximizing the overall "happiness" of everyone in the cloud ecosystem.

We simultaneously target the following goals in our cloud auction design. First, we require the cloud auction to be computationally efficient and execute in polynomial time. Second, the auction should be truthful, so that bidding true job valuation is the dominant strategy for a cloud user. Third, the auction should maximize the social welfare of everyone in the system, including both the cloud provider and the cloud users. Such cloud auction design is faced with a number of challenges. First, truthfulness is a rather strong property that comes only with a pair of carefully prepared VM allocation and payment algorithms that work in concert with each other. Furthermore, even if the cloud users can be assumed to be altruistic and truthful bids are given for free, the winner determination problem for social welfare maximization is an integer linear program (ILP) that is NP-hard to solve. A new challenge unique to our problem here is the non-traditional type of soft deadline constraints, which is hard to model and handle with traditional LP formu-

lation and algorithm design. Last but not least, we require the auction to be online, immediately making a decision upon the arrival of each bid, without knowing future bids in the market, yet still guaranteeing near-optimal decision making as compared to an omniscient offline optimum.

We first consider a basic setting where resources in the cloud are free up to a known capacity limit, and that the soft deadline can be expressed by enumerating a few hard deadline options and their corresponding bidding prices. We first present a natural ILP formulation of the social welfare maximization problem. While polynomial in size, this ILP involves both conventional constraints (capacity limits and XOR bidding) and unconventional constraints (job deadlines). The latter further lead to unconventional dual variables that are hard to interpret and update in a primal-dual algorithm framework we will leverage. We convert the natural ILP into a *compact-exponential* ILP that has a compact formulation of conventional constraints only, at the price of involving an exponential number of variables.

We apply the *posted pricing* primal-dual framework to the compact-exponential ILP for online social welfare maximization. Although the dual has an exponential number of constraints, we show fast dual oracles that can quickly update the dual variables, which are interpreted as unit cost of cloud resources in different time slots. We maintain carefully estimated resource costs based on recently designed exponential cost functions [23]. Upon receiving a bid, we compare the bidding price with the estimated cost of the bid. If the bidding price is higher, the bid is accepted and dual variables are updated; otherwise the bid is rejected. The posted pricing framework charges winning jobs an estimated cost that is independent from the bidding price, and is truthful [42]. We conduct theoretical analysis on the competitive ratio and prove its upper-bound.

We proceed to generalize our cloud auction design by addressing two practical concerns. First, we model the cost of resource provisioning in data centers, using a convex cost function that characterizes server cost with Dynamic Voltage Frequency Scaling [66]. Second, we consider the general form of a soft deadline, specified by (i) a preferred deadline and (ii) a non-decreasing penalty function for deadline violation. The new social welfare maximization problem is an integer

convex program. We resort to a new primal-dual solution framework for well-structured convex programs based on Fenchel dual [31], and adapt our posted pricing auction framework from the previous scenario to this general setting.

In the rest of this chapter, we discuss related work in Ch. 3.2, and introduce the system model in Ch. 3.3. Design and analysis of the online cloud auctions are presented in Ch. 3.4 and Ch. 3.5. Ch. 3.6 presents simulation studies, and Ch. 3.7 concludes the chapter.

3.2 Related Work

Market mechanism design for cloud computing, particularly auction mechanisms for cloud resource trading, has attracted substantial interest from the research community, with a large number of VM auctions spawned in the past few years [62, 64, 75, 77, 78, 80].

The earliest VM auctions are simple in that they are one-round auctions, and assume that the cloud provisions a single type of VM, or that VM configurations are equivalent up to linear scaling [64]. They also assume the scenario of static VM provisioning, where the number and type of VMs to be sold are predetermined prior to the auction start [75].

Dynamic VM provisioning, in which the cloud provider makes decisions on which VMs to assemble and how many based on demand learned from user bids during the auction, has been studied in the past two years [62, 78, 80]. Zhang *et al.* design a randomized auction for dynamic resource provisioning in cloud computing based on a convex decomposition technique, which is truthful and guarantees a small approximation ratio in social welfare [78]. Shi *et al.* further study dynamic resource provisioning where cloud users are subject to budget constraints, and design online auctions where decision making is coupled in the time domain due to fixed user budgets [62].

Online cloud auctions appear later than their one-round counterparts. Zhang *et al.* is among the first to study online cloud auction design, but they assume all VMs are of a uniform type [77]. The work of Shi *et al.* [62] designs online auctions, but does not consider the temporal correlation in decision making due to jobs spanning multiple time slots. A recent work of Zhang *et al.* [80]

studies online cloud auctions where a user bids into a fixed time window for job execution; hence the scheduling dimension is non-present in their solution space.

There have been recent studies on mechanism deign for batch jobs with deadlines. Lucier *et al.* study two scheduling algorithms for jobs with deadlines in cloud computing clusters [53]. They analyze the competitive ratio for non-committed scheduling, which does not require to finish executing a job that has started execution. They do not provide any performance guarantee on the competitive ratio for committed scheduling. Navendu *et al.* design a truthful allocation and pricing mechanism for computing jobs with deadlines, but restrict attention to the offline setting [44]. Azar *et al.* construct an online mechanism for preemptive scheduling with deadlines [18]. Their mechanism is truthful and achieves a constant competitive ratio. All of those works consider only one fixed deadline for each job, and fail to model the server's operation cost.

Compared with existing literature on cloud auctions, our work is the first to design cloud auctions that explicitly consider job elasticity and job execution deadlines, which are important for practical applications to batch processing jobs. Accordingly, we propose the compact-exponential optimization technique that can effectively handle the new job deadline constraints in social welfare maximization for the cloud.

The online primal dual method (see [24] for a detailed survey) is a powerful algorithmic technique that has witnessed broad applications, such as solving the ski rental problem, maximizing revenue in ad-auctions, and solving the general packing problem. The original primal dual framework works on linear programs, and is not used to solve problems modelled by convex programs in our work. Rather recently, new techniques were introduced to help apply the primal dual framework to algorithm design for convex programs. Blum *et al.* study online combinatorial auctions with production costs using the online primal dual framework [20]. They present algorithms for various cost functions. Huang *et al.* further investigate the same problem and propose mechanisms with improved competitive ratio [42]. These studies do not consider the scheduling of jobs, and they cannot handle VM departures and resource recycling.

3.3 System Model

We consider a cloud data center hosting a pool of K types of resources, including CPU, RAM and disk storage that can be dynamically assembled into different types of VMs. Let $[X]$ denote the integer set $\{1,2,\ldots,X\}$. There are a total c_k unit of type-k resource in this cloud. The cloud service provider acts as the auctioneer to lease VMs to cloud users through an auction. User bids arrive sequentially across a large time span $1,2,\ldots,T$. Note that multiple bids can arrive simultaneously, and would be ordered randomly. There are I users participating in the auction, and each user requests customized VMs, and specifies in its bid: (i) r_i^k, the total amount of type-k resource, and (ii) w_i, the number of slots required to finish the job by the designated VMs. Job execution doesn't need to be continuous. A user i's job can be executed at any time slot as long as the total execution time meets w_i before the deadline. We consider two soft deadline models in this work: a basic model with alternative deadlines and a general model with penalty function and server operation cost.

3.3.1 Jobs with Alternative Deadlines

We first consider a basic scenario where each user submits J optional bids to express disjunctive deadline options. A bid from user i consists of a list of desired types of resource r_i^k, $\forall k$; the number of requested slots w_i, and deadlines for job completion d_{ij}, $\forall j$, each with a corresponding bidding price b_{ij}. We use B_i to denote the bidding language of user i's bids submitted at time t_i:

$$B_i = \{t_i, \{r_i^k\}_{k\in[K]}, w_i, \{d_{ij}, b_{ij}\}_{j\in[J]}\}.$$

We adopt the XOR bidding rule that assumes a user can win at most one bid among its J optional bids [78]. Upon the arrival of each bid, the cloud provider decides immediately whether to accept it, and if so, which deadline to choose and how to schedule the job. A binary x_{ij} equals 1 if user i's jth bid wins, and 0 otherwise. Let another binary variable $y_i(t)$ encode the scheduling of user i's job: $y_i(t) = 1$ if user i's job is scheduled to run at time t, and 0 otherwise. The cloud provider also calculates the payment p_i for each winner i.

Let v_{ij} be the true valuation of user i's jth bid, then the utility of that bid is $u_{ij}(b_{ij}) = v_{ij} - p_i$ if $x_{ij} = 1$, and is 0 if $x_{ij} = 0$. In practice, users are assumed to be selfish with a natural goal to maximize their own utilities; they may lie about their true valuations in the hope of a higher utility. The cloud provider instead pursues highest social welfare possible to make everyone in the cloud system "happy". For that goal, it is important for the cloud provider to elicit truthful bids.

Definition 8. (Truthful Auction): A cloud auction is *truthful* if the dominant strategy for each user is to report its true valuation, which always maximizes its utility: for all $b_{ij} \neq v_{ij}$, $u_{ij}(v_{ij}) \geq u_{ij}(b_{ij})$.

Definition 9. (Social Welfare): The social welfare in the cloud market with alternative deadlines is the aggregate user utility $\sum_{i\in[I]}\sum_{j\in[J]} v_{ij}x_{ij} - \sum_{i\in[I]} p_i$ plus the cloud provider's utility $\sum_{i\in[I]} p_i$. Payments cancel themselves, and the social welfare becomes $\sum_{i\in[I]}\sum_{j\in[J]} v_{ij}x_{ij}$.

3.3.2 Jobs with Penalty Function and Operation Cost

We further consider a more general model where each user submits a single preferred deadline d_i, with a penalty function $g_i(\tau_i)$ defined over deadline violation τ_i:

$$g_i(\tau_i) = \begin{cases} g_{c_i}(\tau_i), & \text{if } \tau \in [0, T - d_i] \\ +\infty, & \text{otherwise} \end{cases} \tag{3.1}$$

where $d_i + \tau_i$ is the job completion time; $b_i - g_i(\tau_i)$ is the bidding price, decreasing with job completion time; $g_{c_i}(\cdot)$ is a non-decreasing function and $g_{c_i}(0) = 0$. User i's bid with this model is: $B_i = \{t_i, \{r_i^k\}_{k\in[K]}, w_i, d_i, b_i, g_i(\tau_i)\}$.

Existing studies on cloud auction design often ignore the server operation cost of the cloud provider. It is natural to include server cost in the computation of social welfare, albeit the fact that it makes social welfare optimization substantially more challenging (from linear to non-linear integer programming). The operation cost in the cloud comprises mainly of power consumption for provisioning the virtual machines, increasing as the amount of resources used grows. Let $z_k(t)$ be the amount of type-k resource used at time t in the cloud, then the cost function of type-k resource

is defined as:

$$f_k(z_k(t)) = \begin{cases} \beta_k z_k(t)^{1+\gamma_k}, & \text{if } z_k(t) \in [0, c_k] \\ +\infty, & \text{otherwise} \end{cases} \tag{3.2}$$

Parameter β_k is the coefficient determined by the power consumption of each type of resource. Recent measurement studies suggest that the power consumption of memory, and disk are significantly lower than that of CPU [45]. $\gamma_k \geq 0$ modulates the shape of the cost function, following the operational model of physical servers in the cloud. For example, Dynamic Voltage Frequency Scaling (DVFS) is a technique widely adopted in virtualization platforms, adjusting the frequency or voltage of CPUs to save power consumption [66]. γ_k is roughly 2 if the voltage is proportional to the usage of CPU when DVFS is enabled, and equals 0 when DVFS is disabled [47]. The shape of RAM and disk cost function is different from that of CPU, with $\gamma_k \in [0.5, 1]$ [45].

Similar to the notations in Ch. 3.3.1, let a binary x_i be an auction decision and p_i be the payment. $v_i - g_i{}'(\tau_i)$ is the true valuation of user i's bid. The cloud provider's utility equals the aggregate user payment minus the operation cost, *i.e.,* $\sum_{i\in[I]} p_i - \sum_{k\in[K]} \sum_{t\in[T]} f_k(z_k(t))$. The definitions of user i's utility, truthful auction and social welfare are omitted here as similar ones can be found in Ch. 3.3.1. Table 3.1 summarizes notation for ease of reference.

Table 3.1: Summary of Notation in Chapter 3

I	# of users	$[X]$	integer set $\{1,\ldots,X\}$	T	# of time slots	J	# of bids per user
f	cost function	f^*	convex conjugate of f	g	penalty function	t_i	user i's arrival time

$d_{ij}(d_i)$	deadline of user i's jth (user i's) bid	r_i^k	demand of type-k resource by user i
$b_{ij}(b_i)$	bidding price of user i's jth (user i's) bid	w_i	# of slots requested by user i
$v_{ij}(v_i)$	true valuation of user i's jth (user i's) bid	τ_i	# of slots that passes the deadline for i
$x_{ij}(x_i)$	user i's jth (user i's) bid wins (1) or not (0)	c_k	capacity of type-k resource
$y_i(t)$	whether or not to allocate user i's job in t	u_i	user i's utility
$z_k(t)$	amount of allocated type-k resource at t	p_i	user i's payment
$p_k(t)$	marginal price of type-k resource at t	θ_k	$\max\{2, (1+\gamma_k)^{\frac{1}{\gamma_k}}\}$
$\alpha_1(\alpha_2)$	competitive ratio of $A_{online1}$ ($A_{online2}$)	ρ_k	$\max\{\frac{\theta_k}{c_k}\gamma_k, \frac{\theta_k}{c_k(\theta_k-1)}\ln(\frac{U_k'}{\beta_k(1+\gamma_k)c_k^{\gamma_k}})\}$
$U_k(L_k)$	maximum (minimum) value per unit of type-k resource per unit of time		

3.4 Online Auction Mechanism for Jobs with Alternative Deadlines

In this section, we focus on the scenario where each user's job has J alternative deadlines. Ch. 3.4.1 presents the social welfare maximization problem and the framework to handle such deadline problems. We design an online auction in Ch. 3.4.2 and conduct theoretical analysis in Ch. 3.4.3.

3.4.1 Social Welfare Maximization Problem

Under the assumption of truthful bidding ($b_{ij} = v_{ij}$), the social welfare maximization problem with alternative deadlines can be formulated into the following ILP:

$$\text{maximize} \sum_{i\in[I]} \sum_{j\in[J]} b_{ij} x_{ij} \tag{3.3}$$

$$\text{subject to: } y_i(t)t \leq \sum_{j\in[J]} d_{ij} x_{ij}, \forall t \in [T], \forall i \in [I] : t_i \leq t, \tag{3.3a}$$

$$\sum_{j\in[J]} w_i x_{ij} \leq \sum_{t\in[T]:t_i\leq t} y_i(t), \forall i \in [I], \tag{3.3b}$$

$$\sum_{i\in[I]:t_i\leq t} r_i^k y_i(t) \leq c_k, \forall k \in [K[, \forall t \in [T], \tag{3.3c}$$

$$\sum_{j\in[J]} x_{ij} \leq 1, \forall i \in [I], \tag{3.3d}$$

$$x_{ij}, y_i(t) \in \{0,1\}, \forall i \in [I], \forall t \in [T], \forall j \in [J]. \tag{3.3e}$$

Note that the following constraint is redundant, and is not explicitly included in the ILP above: $y_i(t) \leq \sum_{j\in[J]} x_{ij}, \forall i \in [I], \forall t \in [T]$. Constraint (3.3a) ensures that a job is scheduled to run between its arrival time and deadline. Constraint (3.3b) guarantees that the number of allocated slots is sufficient for serving a successful bid. The capacity limit of each type of resource is expressed in constraint (3.3c), and the alternative deadlines are modelled with the XOR bidding rule by (3.3d).

Even in the offline setting, ILP (3.3) is still an NP-hard combinatorial optimization problem. To verify, consider a special case of ILP (3.3) by setting $T = 1$ and $K = 1$. Then the classic knapsack problem, which is known to be NP-hard, can be reduced to the special case of ILP (3.3) in polynomial time. The challenge further escalates when we involve the jobs' deadlines and pursue online

decision making. To address these challenges, we resort to the primal-dual algorithm design technique. In preparation, we first design a new framework to handle the unconventional constraints for deadline modelling. More specifically, we reformulate the original ILP (3.3) into a simplified *compact-exponential* ILP with a packing structure, at the price of involving an exponential number of variables:

$$\text{maximize} \sum_{i\in[I]} \sum_{l\in\zeta_i} b_{il} x_{il} \tag{3.4}$$

subject to:

$$\sum_{i\in[I]} \sum_{l:t\in T(l)} r_i^k x_{il} \leq c_k, \forall k \in [K], \forall t \in [T], \tag{3.4a}$$

$$\sum_{l\in\zeta_i} x_{il} \leq 1, \forall i \in [I], \tag{3.4b}$$

$$x_{il} \in \{0,1\}, \forall i \in [I], \forall l \in \zeta_i. \tag{3.4c}$$

Constraints (3.4a) and (3.4b) are equivalent to (3.3c) and (3.3d). ζ_i is the set of feasible schedules for user i. A feasible time schedule is a vector $l = \{y_i(t)\}$ that satisfies constraints (3.3a) and (3.3b). Variable $x_{il} \in \{0,1\}$ indicates whether user i's schedule l is accepted (1) or not (0). $T(l)$ records the set of time slots in l. The value of b_{il} is based on schedule l, and equals the corresponding b_{ij}. We relax the integrality constraints of x_{il} to $x_{il} \geq 0$ and formulate the dual problem. By introducing dual variables $p_k(t)$ and u_i to constraints (3.4a) and (3.4b) respectively, the dual LP of the relaxed (3.4) is:

$$\text{minimize} \sum_{i\in[i]} u_i + \sum_{t\in[T]} \sum_{k\in[K]} c_k p_k(t) \tag{3.5}$$

$$\text{subject to: } u_i \geq b_{il} - \sum_{k\in[K]} \sum_{t\in T(l)} r_i^k p_k(t), \forall i \in [I], \forall l \in \zeta_i, \tag{3.5a}$$

$$p_k(t), u_i \geq 0, \forall i \in [I], \forall k \in [K], \forall t \in [T]. \tag{3.5b}$$

As we can observe, a feasible solution to ILP (3.4) has a corresponding feasible solution in ILP (3.3), and the optimal objective value of (3.4) is equal to that of (3.3). The number of variables in ILP (3.4) is exponential since the number of possible time schedules for user i is exponential

in size. We next design an efficient primal-dual allocation scheme that only updates a polynomial number of variables, and can simultaneously solve optimization problems (3.3), (3.4) and (3.5).

3.4.2 Online Auction Design

In the auction algorithm, the cloud provider needs to deicide whether to accept a user i's job and if so, how to schedule its job to meet its deadline. If user i's jth bid with schedule l is accepted, then let $x_{ij} = 1$, and update the variable $y_i(t)$ according to schedule l. To solve ILP (3.3), we adopt the primal-dual technique to the compact-exponential ILP (3.4) and its dual (3.5). For each primal variable x_{il} in (3.4), there is a dual constraint associated with it. Complementary slackness indicates the update of the primal variable is based on its dual constraint. x_{il} is zero unless its associated dual constraint (3.5a) is tight. Because the dual variable $u_i \geq 0$, we let u_i be the maximum of 0 and the right hand side (RHS) of (3.5a),

$$u_i = \max\{0, \max_{l \in \zeta_i}\{b_{il} - \sum_{t \in T(l)} \sum_{k \in [K]} r_i^k p_k(t)\}\}. \tag{3.6}$$

Accordingly, the cloud provider accepts user i if $u_i > 0$, and serves user i's job according to the schedule that maximizes the RHS of constraint (3.5a); if $u_i \leq 0$, the bid is rejected.

Algorithm 1 A Primal-dual Online Auction $A_{online1}$

Input: bidding language $\{B_i\}, \{c_k\}, \{U_k\}, \{L_k\}, \sigma$

Define function $p_k(z_k(t))$ according to (3.7);
Initialize $x_{ij} = 0, y_i(t) = 0, z_k(t) = 0, u_i = 0, p_k(t) = \frac{L_k}{e\sigma}, \forall i \in [I], \forall j \in [J], \forall k \in [K], \forall t \in [T]$;
Let $x_{il} = 0, \forall i \in [I], \forall l \in \zeta_i$, by default;
Upon the arrival of the ith user
$\big(x_{ij}, \{y_i(t)\}, p_i, \{p_k(t)\}, \{z_k(t)\}\big) = A_{core1}\big(B_i, \{c_k\}, \{p_k(t)\}, \{z_k(t)\}\big)$;
if $\exists j \in [J], x_{ij} = 1$ **then**
    Accept user i's jth bid and allocated resources according to $y_i(t)$; Charge p_i for user i;
else
    Reject user i.
end if

If we interpret dual variable $p_k(t)$ as the marginal price per unit of type-k resource at time t, then $\sum_{t \in T(l)} \sum_{k \in [K]} r_i^k p_k(t)$ is the total charge that user i should pay when its job is assigned according to schedule l. The RHS of (3.5a) becomes the utility of bid i with schedule l. Thus, the

Algorithm 2 A Scheduling Algorithm A_{core1}

Input: bidding language $\{B_i\}, \{c_k\}, \{p_k(t)\}, \{z_k(t)\}$
Output: $x_{il}, p_i, \{p_k(t)\}, \{z_k(t)\}$

$c(t) = \sum_{k\in[K]} r_i^k p_k(t), \forall t \in [T]$; // price per slot
for all $j \in [J]$ **do**
    Select w_i slots with minimum $(c(t))$ and $z_k(t) + r_i^k \leq c_k, \forall k \in [K]$ within $[t_i, d_{ij}]$, save the schedule in l_j;
    $p_{ij} = \sum_{t\in T(l)_j} c(t); u_{ij} = b_{ij} - p_{ij}$;
end for
$j^* = \arg\max_{j\in[J]}\{u_{ij}\}$;
if $u_{ij^*} > 0$ **then**
    $x_{ij^*} = 1; y_i(t) = 1, \forall t \in T(l_{j^*}), p_i = p_{ij^*}$;
    $x_{il_{j^*}} = 1$;
    $u_i = u_{ij^*}; z_k(t) = z_k(t) + r_i^k, \forall k \in [K], t \in T(l_{j^*})$;
    $p_k(t) = p_k(z_k(t)), \forall k \in [K], t \in T(l_{j^*})$;
end if
Return $x_{ij^*}, \{y_i(t)\}, p_i, \{p_k(t)\}, \{z_k(t)\}$

assignment of u_i in (3.6) effectively maximizes user i's utility. This is a key step towards achieving social welfare maximization and truthfulness.

Note that although the calculation of u_i seems to take exponential time, since the size of dual constraint (3.5a) is exponential, we design a dual oracle that selects only a polynomial number of dual constraints. We fix a set of schedules L_i with polynomial size through the dual oracle, and set $u_i = \max\{0, \max_{l\in L_i}\{b_{il} - \sum_{t\in T(l)}\sum_{k\in[K]} r_i^k p_k(t)\}\}$. Then x_{il} is updated to 1 when $u_i > 0$. The dual oracle works as follows. For each deadline d_{ij} of user i's job, we select w_i slots with the minimum price for $t \in [t_i, d_{ij}]$, and let l_j be the corresponding schedule, and add l_j to set L_i. The schedule that maximizes user i's utility is the one with the minimum price in set L_i.

We next discuss the update of the dual variable $p_k(t)$. Recall that $p_k(t)$ represents the marginal price per unit of type-k resource at time t. We define a new variable $z_k(t)$ as the amount of allocated type-k resource at time t, and let the marginal price be a function of $z_k(t)$. $p_k(t)$ is increasing with the growth of z_k. Let U_k and L_k be the maximum and minimum values per unit of type-k resource per unit of time, respectively. The initial price of each type-k resource should be low enough such that any user's bid can be accepted. Thus, we set the starting price to $\frac{L_k}{e\sigma}$ where $\sigma > 0$ is a parameter

such that $\frac{T}{\sigma} = \min_{i\in[I]}\{w_i\}$. $p_k(t)$ exponentially increases when $z_k(t)$ is close to the capacity c_k. It reaches U_k when $z_k(t) = c_k$ because in this case, the cloud provider will never allocate any type-k resource to any user. In summary, $p_k(t)$ is defined as a function on $z_k(t)$ as follows:

$$p_k(z_k(t)) = \frac{L_k}{e\sigma}\left(\frac{e\sigma U_k}{L_k}\right)^{\frac{z_k(t)}{c_k}} \tag{3.7}$$

Where $U_k = \max_{i\in[I],j\in[J]:r_i^k>0}\{\frac{b_{ij}}{r_i^k}\}$ and $L_k = \min_{i\in[I],j\in[J]:r_i^k>0}\{\frac{b_{ij}}{w_i\sum_{k\in[K]} r_i^k}\}$.

$A_{online1}$ in Alg. 1 with the scheduling algorithm A_{core1} in Alg. 2 running for each user is the online auction. $A_{online1}$ first defines the price function and initializes the primal and dual variables in lines 1-2. Upon the arrival of each user i, we select the bid j^* with schedule l_{j^*} that maximizes user i's utility through the dual oracle (lines 2-5). If user i obtains positive utility, primal variables x_{ij^*} and $y_i(t)$ are updated according to schedule j^* (line 8). We then increase the usage for different resources ($z_k(t)$) and update the price ($p_k(t)$) for $t \in T(l)_{j^*}$ (lines 10-11).

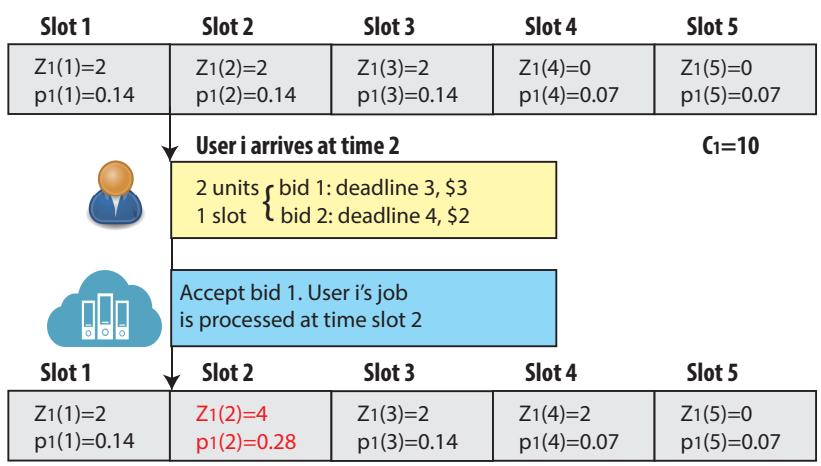

Figure 3.1: An Example of the process in $A_{online1}$.

We next use an example to illustrate the winner determination process in $A_{online1}$, as shown in Fig. 3.1. Suppose the online system spans 5 time slots. A cloud data center hosts only one type of resource and the capacity is 10, *i.e.*, $c_1 = 10$. Assume $\sigma = 5$, $L_1 = 1$ and $U_1 = 2$. Before the arrival of user i, assume the marginal price per unit of resource at time t is $p_1(1) = p_1(2) = p_1(3) \approx 0.14; p_1(4) = p_1(5) \approx 0.07$. The amount of allocated resource at time t is $z_1(1) = z_1(2) = z_1(3) = 2; z_1(4) = z_1(5) = 0$. User i arrives at time 2, requiring 2 units of resource and 1 time slot

to execute its job. It submits two optional bids: it is willing to pay \$3 if its job is completed before time 3 or \$2 if its job is finished before time 4. The bidding price of the user i can be expressed as $B_i = \{2, 2, 1, \{3, \$3\}, \{4, \$2\}\}$. Upon the arrival of the user i, A_{core1} is executed to decide whether to accept it and if so, how to schedule the job. The price per slot is calculated at line 1 in A_{core1} and $c(1) = c(2) = c(3) = 0.28; c(4) = c(5) = 0.14$. For the first bid of user i, lines 3-4 in A_{core1} compute the schedule, payment and utility of it: $l_1 = [2]$, $p_{11} = 0.28$ and $u_{11} = 2.72$. For the second bid of user 1, $l_2 = [4]$, $p_{12} = 0.14$ and $u_{12} = 1.86$. Now user i's maximum utility is larger than 0, *i.e.,* $u_{11} > 0$, primal and dual variables are updated accordingly at lines 8-11 in A_{core1}. Here $z_1(t)$ and $p_1(t)$ at slot 2 are updated, *i.e.,* $z_1(2) = 4$ and $p_1(2) = \frac{1}{5e} \cdot (10e)^{\frac{4}{10}} \approx 0.28$. User i's first bid is accepted and its job is processed at time slot 2. The cloud provider charges \$0.28 for user i. This process is repeated until the last user's job is handled.

3.4.3 Theoretical Analysis

i) Correctness, Running Time, and Truthfulness.

Theorem 1. $A_{online1}$ computes a feasible solution to ILP (3.3), ILP (3.4) and LP (3.5) in $O(IJKT)$ time.

Proof: *(Correctness):* Note that there always exist feasible solutions to ILP (3.3) ILP (3.4) and LP (3.5). Consider variables with initial values (defined in line 2 in $A_{online1}$), they satisfy all constraints and are feasible. $A_{online1}$ outputs a feasible solution for ILP (3.3) because line 3 in A_{core1} guarantees that the schedule l_j for user i's jth bid satisfies constraints (3.3a), (3.3b) and (3.3c). Constraint (3.3d) holds as only one bid per user can be accepted by A_{core1} in line 6. Furthermore, the corresponding relation between x_{ij} and x_{il} implies x_{il} is a feasible solution for ILP (3.4). For the dual problem (3.5), A_{core1} assigns 0 to u_i if $b_{il} \leq \sum_{k\in[K]} \sum_{t\in T(l)} r_i^k p_k(t)$, and $b_{il} - \sum_{k\in[K]} \sum_{t\in T(l)} r_i^k p_k(t)$ to u_i otherwise, ensuring the feasibility of $A_{online1}$.

(Running time): Lines 1-2 can be executed in linear time for the initialization of the cost function, primal and dual variables. Upon the arrival of user i, Algorithm A_{core1} first takes T

steps to calculate the price of each slot. The `for` loop iterates J times to select the best slots for each bid. Line 3 in Alg. 2 takes $O(TK)$ time to schedule the job and check the capacity limit. Line 4 can be done in $O(1)$ steps. Thus, the running time of the `for` loop in Alg. 2 is $O(JKT)$. Then line 6 records the bid with the maximum utility in J steps. The body of the `if` statement (line 8-11) takes $O(KT)$ time to update the primal and dual variables and compute the payment. To sum up, the running time of A_{core1} is $O(JKT)$. The last step of $A_{online1}$ (lines 5-9) is to announce the auction decision, which can be done in constant time. In conclusion, $A_{online1}$ runs in $(O(IJKT))$ time. □

Theorem 2. The online auction $A_{online1}$ is truthful.

Proof: Our auction $A_{online1}$ belongs to the family of *posted pricing mechanisms* [42]. Upon the arrival of user i, the payment that user i needs to pay to the cloud provider, if it wins, depends only on the amount of resource that has been sold, and user i's demand. It is independent of user i's bidding price. Consequently, user i cannot improve its utility by lying about its bidding price since its utility equals its valuation minus the payment, *i.e.,* $u_{ij} = v_{ij} - p_i$. Furthermore, $A_{online1}$ always selects the schedule with the maximum utility among all possible schedules for user i. Hence, truthful bidding guarantees that each user obtains its maximum utility in $A_{online1}$. □

ii) Competitive Ratio.

We next examine the competitive ratio of our online auction. The *competitive ratio* is the upper-bound ratio of the social welfare achieved by the optimal solution of ILP (3.3) to the social welfare achieved by our online auction $A_{online1}$. We first introduce a primal-dual analysis framework in Lemma 1, which states that if there exists a bound between the increase of the primal objective value and the increase of the dual objective value, and the initial dual value is bounded, then the competitive ratio is also bounded. The initial dual value is explained in Lemma 2. We next define the *Allocation-Price Relationship* for $A_{online1}$ in Definition 10 and the differential version of it in Definition 11 respectively. We prove that if the Allocation-Price Relationship holds for a given α_1, $A_{online1}$ satisfies the inequality in Lemma 1. We then present the value of α_1 in Lemma 4 and prove

that $A_{online1}$ is $\frac{e}{e-1}\alpha_1$-competitive in Theorem 3.

Let OPT_1 and OPT_2 denote the optimal objective values of ILP (3.3) and (3.4), respectively. We know that $OPT_1 = OPT_2$. Let P_i and D_i be the objective value of the primal problem (3.4) and that of the dual problem (3.5) returned by an algorithm after processing user i's bids. Let P_0 and D_0 be the initial values. Then P_I and D_I are the final primal and dual objective values achieved by the algorithm.

Lemma 1. If there exists a constant $\alpha_1 \geq 1$ such that i)

$$P_i - P_{i-1} \geq \frac{1}{\alpha_1}(D_i - D_{i-1}), \forall i,$$

ii) $P_0 = 0$ and $D_0 \leq \frac{OPT_2}{e}$, then the algorithm is $\frac{e}{e-1}\alpha_1$-competitive in social welfare.

Proof: When we sum up the inequalities for each i, we have

$$P_I = \sum_i (P_i - P_{i-1}) \geq \frac{1}{\alpha_1}\sum_i (D_i - D_{i-1}) = \frac{1}{\alpha_1}(D_I - D_0).$$

According to weak duality [21], $D_I \geq OPT_2$. Further by our assumption that $D_0 \leq \frac{OPT_2}{e}$. Therefore

$$P_I \geq (1 - \frac{1}{e})\frac{1}{\alpha_1}OPT_2 = (1 - \frac{1}{e})\frac{1}{\alpha_1}OPT_1.$$

So we can conclude that the algorithm is $\frac{e}{e-1}\alpha_1$-competitive. □

Lemma 2. Under the assumption that the offline optimal social welfare is at least $\sum_{k\in[K]} \frac{T}{\sigma} L_k c_k$, *i.e.,* $OPT_2 \geq \sum_{k\in[K]} \frac{T}{\sigma} L_k c_k$, D_0 computed by $A_{online1}$ is at most $\frac{OPT_2}{e}$.

Proof: We first explain the assumption of the lower bound on the offline optimal social welfare. Recall that L_k is the minimum value per unit of type-k resource per unit of time and $\frac{T}{\sigma} = \min_{i\in[I]}\{w_i\}$. $\sum_{k\in[K]} \frac{T}{\sigma} L_k c_k$ is the minimal social welfare generated by bids if the entire capacity of type-k resource is occupied in $\frac{T}{\sigma}$ slots. So the assumption in the above lemma is essentially saying that in the offline solution, there are enough workloads to occupy all types of resources in $\frac{T}{\sigma}$ slots, which is easily satisfied in real-world cloud systems.

The initial objective value of the dual problem (3.5) computed by $A_{online1}$ is:

$$D_0 = \sum_{t\in[T]}\sum_{k\in[K]} c_k p_k(t) = \sum_{t\in[T]}\sum_{k\in[K]} c_k \frac{L_k}{e\sigma}$$
$$= \frac{1}{e}\sum_{k\in[K]} \frac{T}{\sigma} c_k L_k \leq \frac{1}{e} OPT_2.$$

□

$A_{online1}$ guarantees $P_0 = 0$. We next define an *Allocation-Price Relationship* and show that if it holds for a given α_1, then the primal and dual objective values achieved by $A_{online1}$ satisfy the inequality in Lemma 1. $p_k^i(t)$ denotes the price of type-k resource after handling user i. $z_k^i(t)$ is the amount of allocated type-k resource after processing i's job.

Definition 10. The Allocation-Price Relationship for $A_{online1}$ with $\alpha_1 \geq 1$ is:

$$p_k^{i-1}(t)(z_k^i(t) - z_k^{i-1}(t)) \geq \frac{1}{\alpha_1} c_k (p_k^i(t) - p_k^{i-1}(t)), \forall i \in [I], \forall k \in [K], \forall t \in T(l).$$

Lemma 3. If the Allocation-Price Relationship holds for a given $\alpha_1 \geq 1$, then $A_{online1}$ guarantees $P_i - P_{i-1} \geq \frac{1}{\alpha_1}(D_i - D_{i-1})$ for all $i \in [I]$.

Proof: If user i is rejected, then $P_i - P_{i-1} = D_i - D_{i-1} = 0$. In the following analysis, we assume that user i's jth bid is accepted, and let l be the schedule of user i's job. The increment of the primal objective value is: $P_i - P_{i-1} = b_{il}$. Note that $A_{online1}$ makes the constraint (3.5a) tight when bid b_{ij} with schedule l is accepted. Thus,

$$b_{il} = u_i + \sum_{k\in[K]}\sum_{t\in T(l)} p_k^{i-1}(t)(z_k^i(t) - z_k^{i-1}(t)).$$

The increase of the dual objective value is:

$$D_i - D_{i-1} = u_i + \sum_{k\in[K]}\sum_{t\in T(l)} c_k (p_k^i(t) - p_k^{i-1}(t)).$$

By summing up the Allocation-Price Relationship over all $k \in [K]$ and $t \in T(l)$, we can obtain:

$$P_i - P_{i-1} \geq u_i + \frac{1}{\alpha_1}(D_i - D_{i-1} - u_i).$$

Since $u_i \geq 0$ and $\alpha_1 \geq 1$, it is obvious that

$$P_i - P_{i-1} \geq \frac{1}{\alpha_1}(D_i - D_{i-1}).$$

□

We observe that each inequality in the Allocation-Price Relationship involves variables only for type-k resource. Next, we are trying to identify the corresponding $\alpha_{1,k}$ for each pair of k that satisfies the Allocation-Price Relationship. Then α_1 is just the maximum value among all $\alpha_{1,k}$. In order to compute the value of $\alpha_{1,k}$, we make the following mild assumption and define the differential version of the Allocation-Price Relationship based on it.

Assumption 1. The job demand is much smaller than the resource's capacity, *i.e.,* $r_i^k \ll c_k$.

In the real world, a job's demand is usually smaller than a type of resource's capacity in a large data center. We make this assumption mainly to facilitate our theoretical analysis, such that techniques from calculus (differentiation) can be used. We don't consider extreme cases which are rare in practice. For example, if a high-valued bid demanding almost all the resource is rejected, because a small fraction of the resource is used by other users, then the worst-case competitive ratio can be infinitely large. It is also worth noting that, similar assumptions are made in relevant literature of online resource allocation [80] [9] [43]. In addition, we can relax Assumption 1 and assume an upper bound on $\frac{r_i^k}{c_k}$. Instead of differential equation and integration, we can use difference equation and summation to derive similar results. We also relax this assumption completely in our simulation studies. Under Assumption 1, $z_k^i(t) - z_k^{i-1}(t) = dz_k(t)$. The Differential Allocation-Price Relationship is:

Definition 11. The Differential Allocation-Price Relationship for $A_{online1}$ with $\alpha_{1,k} \geq 1$ is

$$p_k(t)dz_k(t) \geq \frac{c_k}{\alpha_{1,k}} dp_k(t), \forall i \in [I], \forall k \in [K], \forall t \in T(l).$$

Lemma 4. $\alpha_{1,k} = \ln\left(\sigma \frac{U_k}{L_k}\right) + 1$ and the marginal price defined in (3.7) satisfies the Differential Allocation-Price Relationship.

Proof: The derivative of the marginal price function is:

$$dp_k(t) = p_k'(z_k(t))dz_k(t) = \frac{L_k}{e\sigma}\left(\frac{e\sigma U_k}{L_k}\right)^{\frac{z_k(t)}{c_k}} \ln\left(\frac{e\sigma U_k}{L_k}\right)^{\frac{1}{c_k}} dz_k(t).$$

The Differential Allocation-Price Relationship is:

$$\begin{aligned}
&\frac{L_k}{e\sigma}\left(\frac{e\sigma U_k}{L_k}\right)^{\frac{z_k(t)}{c_k}} dz_k(t) \geq \frac{c_k}{\alpha_{1,k}} \cdot \frac{L_k}{e\sigma}\left(\frac{e\sigma U_k}{L_k}\right)^{\frac{z_k(t)}{c_k}} \ln\left(\frac{e\sigma U_k}{L_k}\right)^{\frac{1}{c_k}} dz_k(t) \\
&\Rightarrow \alpha_{1,k} \geq \ln\left(\sigma\frac{U_k}{L_k}\right) + 1.
\end{aligned}$$

Therefore this lemma holds for $\alpha_{1,k} = \ln\left(\sigma\frac{U_k}{L_k}\right) + 1$. □

Theorem 3. The online auction $A_{online1}$ in Alg. 1 is $\frac{e}{e-1}\alpha_1$-competitive in social welfare with $\alpha_1 = \max_{k\in[K]}\left\{\ln\left(\sigma\frac{U_k}{L_k}\right)\right\} + 1$.

Proof: According to the proof in Lemma 4, α_1 satisfies the Differential Allocation-Price Relationship. Under Assumption 1, we have $z_k^i(t) - z_k^{i-1}(t) = dz_k(t)$, and

$$dp_k(t) = p_k'(z_k(t))dz_k(t) = p_k^i(t) - p_k^{i-1}(t).$$

As a result, we can show that the Allocation-Price Relationship holds for α_1. Then, combining Lemma 1, Lemma 2 and Lemma 3 we finish the proof. □

3.5 Online Auction Design for the General Model with Penalty Function and Operation Cost

In this section, we present the online auction design for the general model that includes a penalty function and operation cost. We focus on the more challenging case of superlinear cost function with $\gamma_k > 0$. The auction design for linear cost with $\gamma_k = 0$ is similar and is omitted here.

3.5.1 Social Welfare Maximization Problem

Under the assumption of truthful bidding, the social welfare maximization problem in the general model is:

$$\text{maximize} \quad \sum_{i\in[I]}(b_i x_i - g_i(\tau_i)) - \sum_{t\in[T]}\sum_{k\in[K]} f_k(z_k(t)) \tag{3.8}$$

subject to:

$$y_i(t)t \le d_i + \tau_i,\ \forall t \in [T], \forall i \in [I] : t_i \le t, \tag{3.8a}$$

$$w_i x_i \le \sum_{t\in[T]:t_i\le t} y_i(t),\ \forall i \in [I], \tag{3.8b}$$

$$\sum_{i\in[I]:t_i\le t} y_i(t) r_i^k \le z_k(t),\ \forall k \in [K], \forall t \in [T], \tag{3.8c}$$

$$\tau_i, z_k(t) \ge 0, x_i,\ y_i(t) \in \{0,1\},$$
$$\forall i \in [I], \forall t \in [T], \forall k \in [K]. \tag{3.8d}$$

Again, constraint $y_i(t) \le x_i, \forall i,t$ is redundant and is implied by the other constraints. Recall the definition of the cost function in (3.2) ($f_k(z_k(t)) = +\infty$ if $z_k(t) > c_k$), constraint (3.8c) guarantees that the amount of allocated resource never exceeds its capacity.

Let ζ_i be the set of time schedules that satisfy constraints (3.8a) and (3.8b) for user i. We adopt the same framework to reformulate the above convex optimization to the following compact-exponential convex problem:

$$\text{maximize} \quad \sum_{i\in[I]}\sum_{l\in\zeta_i} b_{il} x_{il} - \sum_{t\in[T]}\sum_{k\in[K]} f_k(z_k(t)) \tag{3.9}$$

$$\text{subject to:} \quad \sum_{i\in[I]}\sum_{l:t\in T(l)} r_i^k x_{il} \le z_k(t),\ \forall k \in [K], \forall t \in [T], \tag{3.9a}$$

$$\sum_{l\in\zeta_i} x_{il} \le 1,\ \forall i \in [I], \tag{3.9b}$$

$$x_{il} \in \{0,1\}, z_k(t) \ge 0, \forall i \in [I],\ \forall l \in \zeta_i, \forall k \in [K], \forall t \in [T]. \tag{3.9c}$$

We introduce dual variables $p_k(t)$ and u_i to (3.9a) and (3.9b). The Fenchel dual [31] of the relaxed convex problem (3.9) is:

$$\text{minimize} \sum_{i\in[i]} u_i + \sum_{t\in[T]} \sum_{k\in[K]} f_k^*(p_k(t)) \tag{3.10}$$

$$\text{subject to: } u_i \geq b_{il} - \sum_{k\in[K]} \sum_{t\in T(l)} r_i^k p_k(t), \forall i \in [I], \forall l \in \zeta_i, \tag{3.10a}$$

$$p_k(t), u_i \geq 0, \forall i \in [I], \forall k \in [K], \forall t \in [T]. \tag{3.10b}$$

Here $f_k^*(p_k(t))$ is the convex conjugate [65] of the cost function $f_k(\cdot)$, defined as: $f_k^*(p_k(t)) = \sup_{z_k(t)\geq 0}\{p_k(t)z_k(t) - f_k(z_k(t))\}$.

Lemma 5. The explicit expression of the conjugate is as follows:

$$f_k^*(p_k(t)) = \begin{cases} \left(\dfrac{p_k(t)}{1+\gamma_k}\right)^{\frac{1+\gamma_k}{\gamma_k}} \cdot \dfrac{\gamma_k}{\beta_k^{\frac{1}{\gamma_k}}}, & z_k^0(t) \leq c_k \\ c_k p_k(t) - \beta_k c_k^{1+\gamma_k}, & z_k^0(t) > c_k \end{cases} \tag{3.11}$$

where $z_k^0(t) = (\frac{p_k(t)}{\beta_k(1+\gamma_k)})^{\frac{1}{\gamma_k}}$.

Proof:

$$f_k^*(p_k(t)) = \sup_{z_k(t)\geq 0} \begin{cases} p_k(t)z_k(t) - \beta_k z_k(t)^{1+\gamma_k}, & z_k(t) \in [0, c_k] \\ p_k(t)z_k(t) - \infty, & z_k(t) > c_k \end{cases}$$

We observe that $p_k(t)z_k(t) - \infty = -\infty$ when $z_k(t) > c_k$, thus we only need to obtain the conjugate of f when $z_k(t) \in [0, c_k]$,

Let $\psi_k(z_k(t)) = p_k(t)z_k(t) - \beta_k z_k(t)^{1+\gamma_k}$. The derivative of $\psi_k(z_k(t))$ with respect to $z_k(t)$ is :

$$\psi_k(z_k(t))' = p_k(t) - \beta_k(1+\gamma_k)z_k(t)^{\gamma_k}.$$

When we let $\psi_k(z_k^0(t))' = 0$, the local maximum happens at the point $z_k^0(t)$ and $z_k^0(t) = (\frac{p_k(t)}{\beta_k(1+\gamma_k)})^{\frac{1}{\gamma_k}}$.

Note that the domain of $z_k(t)$ is within the range $[0, c_k]$, therefore the supremum of $\psi_k(z_k(t))$ is $z_k^0(t)$ only if $z_k^0(t) \in [0, c_k]$. Otherwise, when $z_k^0(t) > c_k$, we can obtain that $\psi_k(z_k(t))' > 0$, which means $\psi_k(z_k(t))$ monotonically increases with $z_k(t)$ and the supremum happens at $z_k(t) = c_k$.

To sum up, we derive the conjugate of the cost function as shown in (3.11). □

3.5.2 Online Auction Design

We adopt the same posted pricing primal-dual framework from Ch. 3.4 to solve the convex problem (3.8). Similarly, the primal-dual technique is applied to its compact-exponential problem (3.9) and its dual (3.10): the primal variable x_{il} remains zero unless its dual constraint (3.10a) becomes tight. The assignment of u_i is the same as that in (3.6).

Although there are an exponential number of dual constraints in the computation of u_i, we design a dual oracle based on dynamic programming to output a polynomial number of schedules, then only dual constraints associated to this set of schedules need to be considered.

The basic idea of the dual oracle is as follows. We fix the completion time of user i's job to be t_c ($t_c \in [t_i + w_i - 1, T]$), and construct the best schedule l_j with the minimum price in this case. The set that includes all such l_j has polynomial size, and is the output of the dual oracle. The construction of l_j is based on the dynamic programming method. The base case is the schedule l_0 with $t \in [t_i, t_i + w_i - 1]$. We move the completion time one slot forward each time. Let $c(t) = \sum_{k\in[K]} r_i^k p_k(t)$ be the price of user i's job running at time t. If the completion time t_c passes the deadline d_i, the corresponding penalty is added to the price, *i.e.,* $c(t) = \sum_{k\in[K]} r_i^k p_k(t) + g(t_c - d_i)$. When the old completion time is replaced with the new one, we only need to compare the price of the old completion time and $w_i - 1$ slots before the old completion time. For example, if user i arrives at time 1 with $w_i = 4$, then the basic case is $l_0 = \{1,2,3,4\}$. Assume that $\arg\max_{t\in\{1,2,3\}} c(t) = 2$. We next fix the completion time to 5, the best schedule is then $\{1,4,3,5\}$ if $c(4) < c(2)$ and $\{1,2,3,5\}$ otherwise. The process is repeated until the completion time reaches T.

The marginal price $p_k(t)$ per unit of type-k resource at time t can be defined as the derivative of the cost function, *i.e.,* $f_k{}'(\hat{z_k}(t))$ if the overall demand of resource k at t ($\hat{z_k}(t)$) is known. But in the online setting, it is impossible for the cloud provider to acquire the complete knowledge of the system. The cloud provider predicts the final demand at future slots as $\theta_k (\theta_k > 1)$ times the current demand at those slots if the predicted final demand is below the capacity, and set the marginal price to $f_k{}'(\theta_k z_k(t))$ where $z_k(t)$ is the amount of current allocated resource k at t. Let

U'_k be the maximum value per unit of type-k resource per unit of time. The marginal price grows exponentially when the predicted demand is larger than the capacity, and reaches U'_k if $z_k(t) = c_k$. More specifically, the marginal price function is defined as:

$$p_k(z_k(t)) = \begin{cases} f'_k(\theta_k z_k(t)), & z_k(t) \leq \dfrac{c_k}{\theta_k} \\ f'_k(c_k) e^{\rho_k (z_k(t) - \frac{c_k}{\theta_k})}, & z_k(t) > \dfrac{c_k}{\theta_k} \end{cases} \tag{3.12}$$

$$\text{with parameters } \theta_k = \max\{2, (1+\gamma_k)^{\frac{1}{\gamma_k}}\},$$
$$\rho_k = \max\{\frac{\theta_k}{c_k}\gamma_k, \frac{\theta_k}{c_k(\theta_k - 1)} \ln(\frac{U'_k}{\beta_k(1+\gamma_k)c_k^{\gamma_k}})\},$$

where $U'_k = \max_{i \in [I]: r_i^k > 0}\{\frac{b_i}{r_i^k}\}$.

Algorithm 3 A Primal-dual Online Auction $A_{online2}$

Input: bidding language $\{B_i\}, \{c_k\}, \{\beta_k, \gamma_k\}$

1: Define cost function $f_k(z_k(t))$ according to (3.2);
2: Define function $p_k(z_k(t))$ according to (3.12);
3: Initialize $x_i = 0, y_i(t) = 0, z_k(t) = 0, \tau_i = 0, u_i = 0, p_k(t) = 0, \forall i \in [I], \forall k \in [K], \forall t \in [T]$; Let $x_{il} = 0, \forall i \in [I], \forall l \in \zeta_i$, by default;
4: **Upon the arrival of the** i**th user**
5: $\big(x_i, \{y_i(t)\}, p_i, \{p_k(t)\}, \{z_k(t)\}\big) = A_{core2}\big(B_i, \{c_k\}, \{p_k(t)\}, \{z_k(t)\}\big)$;
6: **if** $x_i = 1$ **then**
7: Accept user i's bid and allocated resources according to $y_i(t)$; Charge p_i for user i;
8: **else**
9: Reject user i.
10: **end if**

The online auction $A_{online2}$ for the general model is presented in Alg. 3. Upon the arrival of the ith user, $A_{online2}$ calls A_{core2} in Alg. 4 to make decisions. A_{core2} computes the best schedule for user i through the dual oracle (lines 1-10) to maximize its utility u_i. If $u_i > 0$, the corresponding primal and dual variables are updated in lines 14-17.

3.5.3 Theoretical Analysis

i) Correctness, Running Time, and Truthfulness.

Lemma 6. The running time of A_{core2} is $O(KT + T^2)$.

Proof: Line 1 initializes a feasible slot set $\mathscr{T}$ in $O(KT)$ steps. Line 2 takes w_i steps to define a schedule l_o The `while` loop (lines 3-11) is to compute the best schedule l_j if the completion time is fixed, will iterate at most $T - w_i$ times. Within the `while` loop body, lines 4-7 takes $O(w_i + 1)$ steps to update $c(t)$. The running time of finding the maximum price in line 8 is linear to w_i. Lines 9-10 takes constant time for the comparison and addition. Thus, the `while` loop can be executed in $O((T - w_i)w_i)$ steps. Line 12 can be done in $O(T - w_i)$ steps to find the schedule with the minimum price. The running time to execute the `if` body is $O(KT)$. In summary, the running time of A_{core2} is $O(KT + T^2)$. □

Algorithm 4 A Scheduling Algorithm A_{core2}.

Input: $B_i, \{c_k\}, \{p_k(t)\}, \{z_k(t)\}$
Output: $x_i, \{y_i(t)\}, p_i, \{p_k(t)\}, \{z_k(t)\}$

1: Add slot $t \in [t_i, T]$ to set $\mathscr{T}$ if $z_k(t) + r_i^k \leq c_k, \forall k \in [K]$;
2: Let schedule l_0 include the first w_i slots $(t_1, t_2, \ldots, t_{w_i})$ in $\mathscr{T}$; Define $j = 1$;
3: **while** $w_i + j \leq |\mathscr{T}|$ **do**
4: $\quad l_j = l_{j-1}$;
5: $\quad$ Let t_c is the $(w_i + j)$th slot in $\mathscr{T}$;
6: $\quad c(t) = \sum_{k \in [K]} r_i^k p_k(t), \forall t \in \{t_1, t_2, \ldots, t_{w_i}, t_c\}$;
7: $\quad$ If $t_c > d_i$, $c(t_c) = c(t_c) + g(t_c - d_i)$;
8: $\quad t_m = \arg\max_{t \in \{t_1, \ldots, t_{w_i - 1}\}} c(t)$;
9: $\quad$ If $c(t_{w_i}) < c(t_m)$, for schedule l_j, replace the slot t_m with t_{w_i} and save t_c into t_{w_i};
10: $\quad \mathscr{P}_j = \sum_{t \in T(l)_j} c(t)$; $j = j + 1$;
11: **end while**
12: $j^* = \arg\min_j \{\mathscr{P}_j\}$;
13: **if** $b_i - \mathscr{P}_{j^*} > 0$ **then**
14: $\quad x_i = 1; y_i(t) = 1, \forall t \in T(l)_{j^*}; x_{il_{j^*}} = 1$;
15: $\quad u_i = b_i - \mathscr{P}_{j^*}; p_i = \sum_{k \in [K]} \sum_{t \in T(l)_{j^*}} r_i^k p_k(t)$;
16: $\quad z_k(t) = z_k(t) + r_i^k, \forall k \in [K], t \in T(l)_{j^*}$;
17: $\quad p_k(t) = p_k(z_k(t)), \forall k \in [K], t \in T(l)_{j^*}$;
18: **end if**
19: **Return** $x_i, \{y_i(t)\}, p_i, \{p_k(t)\}, \{z_k(t)\}$

Theorem 4. $A_{online2}$ in Alg. 3 is a truthful auction that returns feasible solutions for convex problems (3.8), (3.9) and (3.10) in $O(I(KT + T^2))$ time.

Proof: (*Running time:*) The running time of the initialization process in lines 1-3 is linear. By Lemma 6, A_{core2} in line 5 processes each user in $O(KT + T^2)$ time. The `If` statement in lines 6-10

can be done within constant time. Therefore, after handling the last user, the overall running time of $A_{online2}$ is $O(I(KT+T^2))$.

(*Correctness and Truthfulness:*) We omit the proof here since similar proofs can be found in Theorem 1 and Theorem 2. □

ii) Competitive Ratio.

The proof follows the same structure as that in Ch. 3.4.3. Let OPT_{1^*} and OPT_{2^*} denote the optimal objective values of ILP (3.8) and (3.9), respectively. We know that $OPT_{1^*} = OPT_{2^*}$. Let P_i and D_i be the primal (3.9) and dual (3.10) objective values achieved by $A_{online2}$ after handling user i's request. By $A_{online2}$, initial values equal zero, *i.e.,* $P_0 = D_0 = 0$. P_I and D_I are the final primal and dual objective values at the end of T. We first prove that $A_{online2}$ is α_2-competitive in social welfare if there is a constant $a_2 \geq 1$ such that $P_i - P_{i-1} \geq \frac{1}{\alpha_2}(D_i - D_{i-1})$ for all i in Lemma 7. We next define the Allocation-Price Relationship for $A_{online2}$, and show that if the Allocation-Price Relationship holds for a given α_2, then $P_i - P_{i-1} \geq \frac{1}{\alpha_2}(D_i - D_{i-1})$ also holds in Lemma 8. The last step is to define the differential version of the Allocation-Price Relationship and prove in Lemma 9 that there exists a $\alpha_{2,k}$ that satisfies this relationship . By setting $\alpha_2 = \max_{k\in[K]}\{\alpha_{2,k}\}$, we can obtain the competitive ratio of $A_{online2}$ in Theorem 5.

Lemma 7. If there exists a constant $\alpha_2 \geq 1$ such that

$$P_i - P_{i-1} \geq \frac{1}{\alpha_2}(D_i - D_{i-1})$$

for every i, then $A_{online2}$ is α_2-competitive in social welfare.

Proof: If we sum up the inequality for each i, we can obtain,

$$P_I = \sum_i (P_i - P_{i-1}) \geq \frac{1}{\alpha_2}\sum_i (D_i - D_{i-1}) = \frac{1}{\alpha_2} D_I.$$

The above inequality holds because $P_0 = D_0 = 0$. By weak duality [21], $D_I \geq OPT_{2^*}$, therefore

$$P_I \geq \frac{1}{\alpha_2} OPT_{2^*} = \frac{1}{\alpha_2} OPT_{1^*}.$$

So we can conclude that $A_{online2}$ is α_2-competitive in social welfare. □

Definition 12. The Allocation-Price Relationship for $A_{online2}$ with $\alpha_2 \geq 1$ is:

$$p_k^{i-1}(t)\big(z_k^i(t)-z_k^{i-1}(t)\big)-\big(f_k(z_k^i(t))-f_k(z_k^{i-1}(t))\big) \geq \frac{1}{\alpha_2}\big(f_k^*(p_k^i(t))-f_k^*(p_k^{i-1}(t))\big), \forall i, \forall k, \forall t \in T(l).$$

Lemma 8. If the Allocation-Price Relationship for $A_{online2}$ holds with a given $\alpha_2 \geq 1$, then $A_{online2}$ guarantees $P_i - P_{i-1} \geq \frac{1}{\alpha_2}(D_i - D_{i-1})$ for all $i \in [I]$.

Proof: If user i is rejected, then $P_i - P_{i-1} = D_i - D_{i-1} = 0$. In the next analysis, we assume that user i is accepted, and let l be the schedule of user i's job. The increment of the primal objective value is:

$$\begin{aligned} P_i - P_{i-1} &= b_{il} - \sum_{t \in T(l)} \sum_{k \in [K]} \big(f_k(z_k^i(t)) - f_k(z_k^{i-1}(t))\big) \\ &= u_i + \sum_{k \in [K]} \sum_{t \in T(l)} p_k^{i-1}(t)\big(z_k^i(t) - z_k^{i-1}(t)\big) \\ &- \sum_{t \in T(l)} \sum_{k \in [K]} \big(f_k(z_k^i(t)) - f_k(z_k^{i-1}(t))\big). \end{aligned}$$

The second equality holds because $A_{online2}$ updates the value of dual variables such that the dual constraint becomes tight and $r_i^k = z_k^i(t) - z_k^{i-1}(t)$. Then the increase of the dual objective value is:

$$D_i - D_{i-1} = u_i + \sum_{t \in T(l)} \sum_{k \in [K]} \big(f_k^*(p_k^i(t)) - f_k^*(p_k^{i-1}(t))\big)$$

By summing up the Allocation-Price Relationship for $A_{online2}$ over all $k \in [K]$ and $t \in T(l)$, we can obtain:

$$P_i - P_{i-1} \geq u_i + \frac{1}{\alpha_2}(D_i - D_{i-1} - u_i).$$

Since $u_i \geq 0$ and $\alpha_1 \geq 0$, it is obvious that $P_i - P_{i-1} \geq \frac{1}{\alpha_2}(D_i - D_{i-1})$. □

Definition 13. The Differential Allocation-Price Relationship for $A_{online2}$ with $\alpha_{2,k} \geq 1$ is:

$$p_k(t)dz_k(t) - f_k'(z_k(t))dz_k(t) \geq \frac{1}{\alpha_{2,k}} f_k^{*\prime}(p_k(t))dp_k(t), \forall i, \forall k, \forall t \in T(l).$$

Lemma 9. $\alpha_{2,k} = \max\{4(1+\gamma_k), \frac{2(1+\gamma_k)}{\gamma_k}\ln(\frac{U_k'}{\beta_k(1+\gamma_k)c_k^{\gamma_k}})\}$ and the marginal price function defined in (3.12) satisfy the Differential Allocation-Price Relationship.

Proof: We first write down the explicit expressions for the derivatives of the cost function (3.2) and its convex conjugate (3.11):

$$f_k'(z_k(t)) = \begin{cases} \beta_k(1+\gamma_k)z_k(t)^{\gamma_k}, & \text{if } z_k(t) \in [0, c_k] \\ +\infty, & \text{otherwise} \end{cases}$$

$$f_k^{*\prime}(z_k(t)) = \begin{cases} \Big(\dfrac{p_k(t)}{\beta_k(1+\gamma_k)}\Big)^{\frac{1}{\gamma_k}}, & p_k(t) \le \beta_k(1+\gamma_k)c_k^{\gamma_k} \\ c_k, & p_k(t) > \beta_k(1+\gamma_k)c_k^{\gamma_k} \end{cases}$$

When the amount of allocated type-k resource reaches the capacity, *i.e.,* $z_k(t) = c_k$, according to the definition of marginal price in (3.12),

$$p_k(t) = \beta_k(1+\gamma_k)c_k^{\gamma_k} e^{\rho_k(c_k - \frac{c_k}{\theta_k})} \ge U_k'.$$

Recall that U_k' is the maximum value per unit of resource k per unit of time. It is clear when the marginal price is larger than U_k', no bids can win. Thus, we may assume $z_k(t) \le c_k$ in the rest of the proof, and $f_k'(z_k(t)) = \beta_k(1+\gamma_k)z_k(t)^{\gamma_k}$. Next, we divide our proof into two cases:

Case 1: $z_k(t) \le \frac{c_k}{\theta_k}$: Because

$$p_k(t) = f'(\theta_k z_k(t)) = \beta_k(1+\gamma_k)(\theta_k z_k(t))^{\gamma_k} \le \beta_k(1+\gamma_k)c_k^{\gamma_k},$$

the Differential Allocation-Price Relationship can be rewritten as:

$$\begin{aligned} &(\beta_k(1+\gamma_k)(\theta_k z_k(t))^{\gamma_k} - \beta_k(1+\gamma_k)z_k(t)^{\gamma_k})dz_k(t) \\ &\ge \frac{1}{\alpha_{2,k}}\Big(\frac{p_k(t)}{\beta_k(1+\gamma_k)}\Big)^{\frac{1}{\gamma_k}}\beta_k(1+\gamma_k)\theta_k^{\gamma_k}\gamma_k z_k(t)^{\gamma_k-1}dz_k(t). \end{aligned} \tag{3.13}$$

Canceling the common term on both sides, (3.13) becomes $(\theta_k^{\gamma_k} - 1) \ge \frac{1}{\alpha_{2,k}}\gamma_k\theta^{\gamma_k+1}$.

i) If $\gamma_k \ge 1$, $\theta_k = \max\{2, (1+\gamma_k)^{\frac{1}{\gamma_k}}\} = 2$, we can obtain

$$\frac{\gamma_k\theta_k^{\gamma_k+1}}{\theta_k^{\gamma}-1} = \frac{\gamma_k 2\cdot 2^{\gamma_k}}{2^{\gamma}-1} = \frac{\gamma_k(4\cdot 2^{\gamma_k} - 2\cdot 2^{\gamma_k})}{2^{\gamma_k}-1} \le \frac{4\gamma_k(2^{\gamma_k}-1)}{2^{\gamma_k}-1} \le 4\gamma_k < \alpha_{2,k}$$

ii) If $\gamma_k < 1$, then $\theta_k = (1+\gamma_k)^{\frac{1}{\gamma_k}} < e$, and

$$\frac{\gamma_k \theta_k^{\gamma_k+1}}{\theta_k^{\gamma} - 1} = \theta_k(1+\gamma_k) < e(1+\gamma_k) < \alpha_{2,k}.$$

Case 2: $z_k(t) > \frac{c_k}{\theta_k}$: In this case, the marginal price $z_k(t)$ is:

$$p_k(t) = \beta_k(1+\gamma_k)c_k^{\gamma_k} e^{\rho_k(z_k(t) - \frac{c_k}{\theta_k})}.$$

Note that $dp_k(t) = \rho_k p_k(t) dz_k(t)$, then the Differential Allocation-Price Relationship is:

$$(p_k(t) - f_k'(z_k(t)))dz_k(t) \geq \frac{1}{\alpha_{2,k}} c_k \rho_k p_k(t) dz_k(t). \tag{3.14}$$

By Lemma 10, we can obtain

$$p_k(t) - f_k'(z_k(t)) \geq p_k(t) - \frac{1}{1+\gamma_k} p_k(t) \geq \frac{\gamma_k}{1+\gamma_k} p_k(t),$$

thus to prove (3.14), it is sufficient to prove:

$$\frac{\gamma_k}{1+\gamma_k} p_k(t) dz_k(t) \geq \frac{1}{\alpha_{2,k}} c_k \rho_k p_k(t) dz_k(t) \Rightarrow \rho_k \leq \frac{\gamma_k}{c_k(1+\gamma_k)} \alpha_{2,k}.$$

By the value of ρ_k, either **i)**

$$\rho_k = \frac{\theta_k}{c_k}\gamma_k \leq \frac{e}{c_k}\gamma_k = \frac{\gamma_k}{c_k(1+\gamma_k)} e(1+\gamma_k) \leq \frac{\gamma_k}{c_k(1+\gamma_k)} \alpha_{2,k}.$$

$$\begin{aligned} \text{Or } \textbf{ii)}\ \rho_k &= \frac{\theta_k}{c_k(\theta_k - 1)} \ln\left(\frac{U_k'}{\beta_k(1+\gamma_k)c_k^{\gamma_k}}\right) \leq \frac{2}{c_k} \ln\left(\frac{U_k'}{\beta_k(1+\gamma_k)c_k^{\gamma_k}}\right) \\ &= \frac{\gamma_k}{c_k(1+\gamma_k)} \frac{2(1+\gamma_k)}{\gamma_k} \ln\left(\frac{U_k'}{\beta_k(1+\gamma_k)c_k^{\gamma_k}}\right) \leq \frac{\gamma_k}{c_k(1+\gamma_k)} \alpha_{2,k}. \end{aligned}$$

In conclusion, we have finished the proof for both cases. □

Lemma 10. When $z_k(t) > \frac{c_k}{\theta_k}$, the marginal price $p_k(t)$ is larger than the marginal cost by a factor of at least $1+\gamma_k$:

$$p_k(t) \geq (1+\gamma_k) f_k'(z_k(t)).$$

Proof: When $z_k(t) > \frac{c_k}{\theta_k}$, $p_k(t) = \beta_k(1+\gamma_k)c_k^{\gamma_k}e^{\rho_k(z_k(t)-\frac{c_k}{\theta_k})}$. So Lemma 10 is equivalent to verify

$$\frac{e^{\rho_k z_k(t)}}{z_k(t)^{\gamma_k}} \geq \frac{(1+\gamma_k)e^{\frac{\rho_k c_k}{\theta_k}}}{c_k^{\gamma_k}} \tag{3.15}$$

We first show that the inequality (3.15) holds when $z_k(t) = \frac{c_k}{\theta_k}$. If $z_k(t)$ takes the value of $\frac{c_k}{\theta_k}$, (3.15) becomes $\theta_k^{\gamma_k} \geq 1+\gamma_k$ which is obviously true.

Next, it suffices to show the left side of (3.15) is non-decreasing as $z_k(t)$ increases. Let $L(z_k(t))$ denote the left hand of (3.15). The derivative of $L(z_k(t))$ is

$$L'(z_k(t)) = \frac{e^{\rho_k z_k(t)}(\rho_k z_k(t) - \gamma_k)}{z_k(t)^{1+\gamma_k}}.$$

Because $\rho_k \geq \frac{\theta_k}{c_k}\gamma_k$ and $z_k(t) > \frac{c_k}{\theta_k}$, then $\rho_k z_k(t) - \gamma_k \geq 0$ and the derivative $L'(z_k(t))$ is non-negative. Consequently, the lemma follows. □

Theorem 5. The online auction $A_{online2}$ in Alg. 3 is α_2-competitive in social welfare with $\alpha_2 = \max_{k\in[K]} \alpha_{2,k}$.

Proof: Because α_2 is the maximum number among all $\alpha_{2,k}$, then Differential Allocation-Price Relationship also holds with α_2. Under Assumption 1, we have $dz_k(t) = z_k^i(t) - z_k^{i-1}(t)$, then

$$f_k(z_k^i(t)) - f_k(z_k^{i-1}(t)) = f_k'(z_k^{i-1}(t))(z_k^i(t) - z_k^{i-1}(t)),$$

$$f_k^*(p_k^i(t)) - f_k^*(p_k^{i-1}(t)) = f_k^{*\prime}(p_k^{i-1}(t))(p_k^i(t) - p_k^{i-1}(t)).$$

Therefore, the Allocation-Price Relationship holds with α_2. Combining Lemma 7 and Lemma 8, we finish the proof. □

We plot the value of α_2 in Fig. 3.2 when we vary the value of γ_k, β_k, c_k and U_k' [45,47]. We can observe that if we normalize c_k to 1, the competitive ratio of $A_{online2}$ is close to 6 with a small U_k' and γ_k, as demonstrated in the left figure. The right figure shows that if c_k is a large number, the competitive ratio is determined by γ_k and increases with the increment of γ_k.

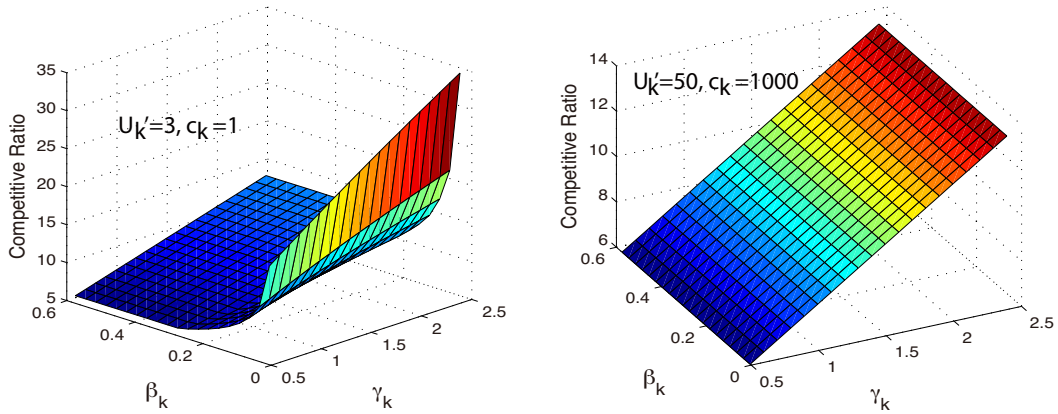

Figure 3.2: An illustration of the competitive ratio of $A_{online2}$ (α_2) under different settings.

3.6 Performance Evaluation

We evaluate our online auctions $A_{online1}$ and $A_{online2}$ through trace-driven simulation studies. We exploit the trace version 1 in Google Cluster Data [3], which contains the information for each job including the start time, execution duration, and resource demands (CPU and RAM). We translate each job into a bid, arriving sequentially in 18 hours. We assume that each user's job consumes $[1,12]$ slots and each time slot is 5 minutes [36]. User's job deadline is generated uniformly at random between its arrival time and the system end time. The bidding price of each job equals its overall resource demand times unit prices randomly picked in the range $[L_k, U_k]$. By default, $L_k = 1$ and $U'_k = U_k = 50$. The demand for CPU and RAM units is normalized so that the maximum capacity is 1. For the cost function, β_k is set within $[0.4, 0.6]$ for CPU and within $[0.005, 0.02]$ for RAM [45]. γ_k is set within $[1.7, 2.2]$ for CPU and within $[0.5, 1]$ for RAM [47].

3.6.1 Performance of $A_{online1}$.

We examine the performance of $A_{online1}$ in terms of the competitive ratio, social welfare and user satisfaction.

Fig. 3.3 shows the competitive ratio of $A_{online1}$ with different numbers of users (I) and bids per

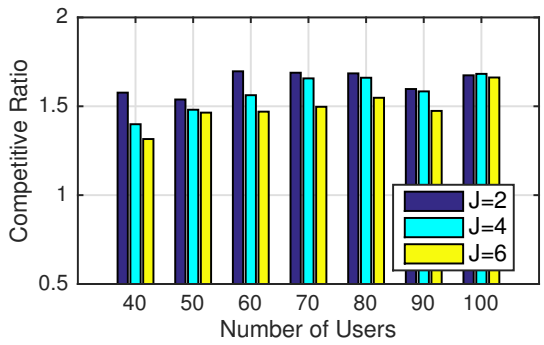

Figure 3.3: Competitive ratio of $A_{online1}$ with different number of users and J.

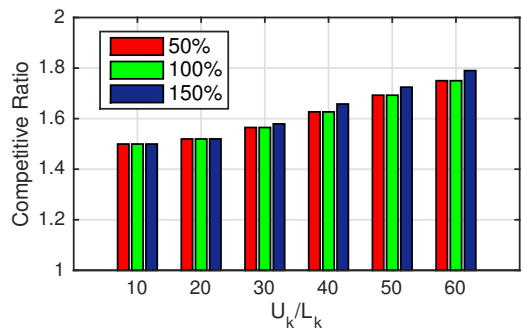

Figure 3.4: Competitive ratio of $A_{online1}$ with different U_k/L_k.

user (J). The observed competitive ratio is much better than the theoretical bound and remains at a low level (< 2). It fluctuates with the increase of the number of users and sightly decreases when the number of bids per user grows. This is because when each user provides a larger number of optional bids, $A_{online1}$ is more likely to optimize the schedule of its job, leading to a better performance. In $A_{online1}$, the marginal price function is defined based on the real value of U_k and L_k. We vary the value of U_k/L_k, and use the estimated values of U_k as the input of $A_{online1}$, to examine the performance. As shown in Fig. 3.4, there is a downward trend as the value of U_k/L_k decreases, while there is no large difference with either underestimation and overestimation. The observation confirms the analysis in Theorem 3 that the value of U_k/L_k determines the competitive ratio. Underestimation is more desirable than the overestimation, as compared to that achieved by the real U_k (labelled by 100%). Overestimation makes the price rise more rapidly, filtering out users that are supposed to be accepted.

We next study the social welfare achieved by $A_{online1}$ in Fig. 3.5 and Fig. 3.6. The 3d figure in Fig. 3.5 plots the social welfare under different number of users and bids per user. Our online auction $A_{online1}$ achieves a higher social welfare when there is larger number of users participating in the auction. The change of bids per user doesn't have major influence on the social welfare. When the number of users grows, the number of bids with larger bidding price also increases. As a result, A_{online} returns a larger social welfare. The social welfare under different number of slots and U_k/L_k is illustrated in Fig. 3.6. Both the number of slots and the value of U_k/L_k influence

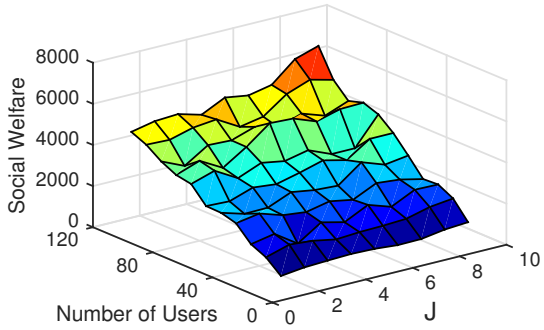

Figure 3.5: Social welfare of $A_{online1}$ with different number of users and J.

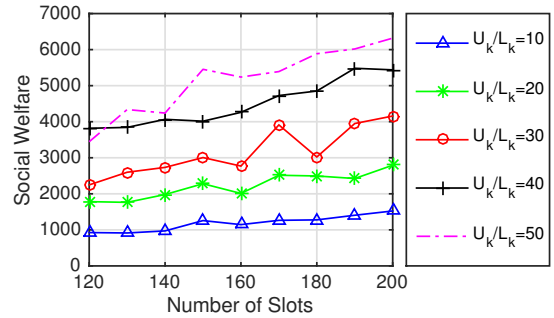

Figure 3.6: Social welfare of $A_{online1}$ with different T and U_k/L_k

the social welfare. $A_{online1}$ is able to allocate more jobs when the length of the system increases. Furthermore, the bidding price rises with the increase of U_k/L_k, thus high value bids lead to a higher social welfare.

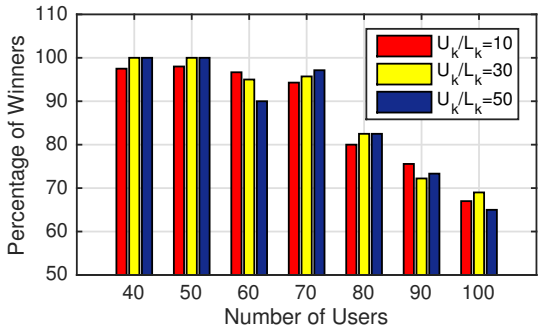

Figure 3.7: Percentage of winners in $A_{online1}$ with different I and U_k/L_k

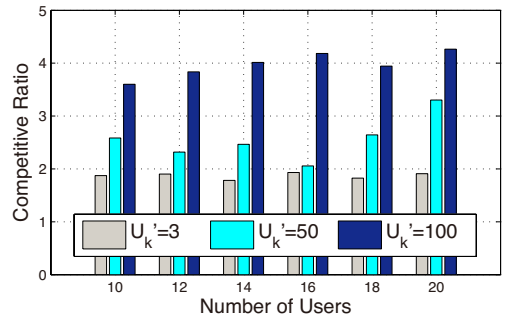

Figure 3.8: Competitive ratio of $A_{online2}$ with different I and U_k'

User satisfaction, which is measured by the percentage of winners, is demonstrated in Fig. 3.7. A higher fraction of users are accepted with a small number of users. This is because the number of winners is almost fixed due to the capacity limit. We also obverse that the value of U_k/L_k doesn't influence the percentage since the winner determination process is not affected by the change of U_k/L_k.

3.6.2 Performance of $A_{online2}$.

We first examine the competitive ratio of $A_{online2}$. We use `CVX` with the Gurobi Optimizer to solve the convex problem (3.8) exactly, and compute the competitive ratio by dividing the optimal social welfare by the social welfare returned by $A_{online2}$. However, `CVX` fails to solve in 24 hours even with a medium-size input. Thus, we reduce the input size and only consider 10-20 users. Fig. 3.8 shows the competitive ratio of $A_{online2}$ under different number of users and U'_k. It becomes larger with the increase of U'_k. The change of the number of users doesn't have much impact on the competitive ratio. As indicated in Theorem 5, a larger U'_k negatively influences the competitive ratio when we set c_k to 1. We can also observe that the competitive ratio is still less than 5 with a large U'_k, which is much better than the theoretical bound.

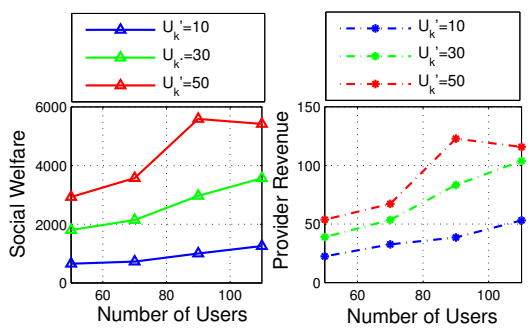

Figure 3.9: Social welfare and cloud provider's revenue in $A_{online2}$ with I and U'_k.

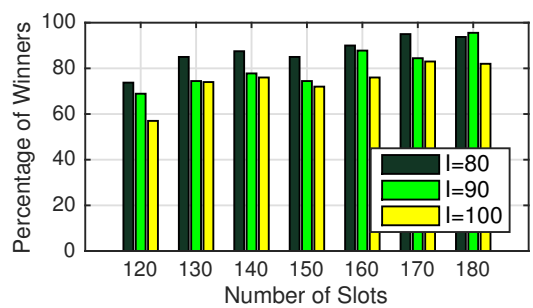

Figure 3.10: Percentage of winners in $A_{online2}$

We next study the performance of $A_{online2}$ in the aspects of social welfare and user satisfaction. Fig. 3.9 shows the social welfare and cloud provider's revenue with different number of users when we vary the value of U'_k. We can observe that both the social welfare and revenue increase with the number of users and U'_k. The reason has been explained when we evaluated the performance of $A_{online1}$ and is omitted here, as the design of $A_{online2}$ follows the same primal-dual technique. Fig. 3.10 shows that the percentage of winners gradually rises when the number of slots increases. The possibility of winning becomes higher when the system spans a long period since there are more slots available for scheduling. In addition, a small number of users leads to higher user satisfaction.

3.7 Summary

In this chapter, we studied the auction design for cloud computing jobs that have soft completion deadlines. Our main contribution is an online cloud job auction that is truthful and computationally efficient, and achieves a good competitive ratio in social welfare. Techniques used in the auction design include the posted pricing framework for truthful online auctions, a new LP formulation and solution method for handling soft deadline constraints, as well as approximation algorithms based on LP dual and Fenchel dual. Our method for handling soft deadline constraints may be applicable to other auction design problems where deadlines are involved, for example, in demand response auctions in a smart grid.

Chapter 4

Scheduling Frameworks for Cloud Container Services

4.1 Introduction

In this chapter, we extend the existing literature in cloud resource provisioning, and propose the first offline and online scheduling frameworks for cloud container provisioning. We simultaneously target the following goals. **First**, we require the schedulers to be time efficient, running in polynomial time. **Second**, the aggregate value of jobs that are completed before their deadlines should be maximized. **Third**, the schedulers permit partial execution and can handle general type of jobs, *i.e.,* jobs with multiple subtasks, defined by i) the dependence graph that captures the dependence among subtasks; ii) the resource profile of each container, which is dedicated to each subtask; iii) the deadline for job completion; iv) the value of each subtask.

We formulate the offline optimization problem into a natural Integer Linear Program (ILP). While polynomial in size, this ILP involves non-conventional scheduling constraints that are hard to handle by the classic primal-dual framework. We apply the *compact-exponential* technique [84] to reformulate the problem into a *compact-exponential ILP*, which is a conventional packing-type ILP with an exponential number of variables corresponding to valid schedules. This compact-exponential ILP and its dual form the foundation of our offline and online scheduling algorithm design. We will show that the substantially amplified ILP size can be managed through the primal-dual technique, for computing a close-to-optimal aggregate job valuation in polynomial time.

We first assume that job information is known in advance, and focus on offline scheduling algorithm design under resource capacity and job scheduling constraints. Besides serving as a benchmark for our online algorithm, the offline algorithm is also applicable to a limited near-future time window for which job information can be predicted. We leverage the classic randomized rounding technique [59]. Given a fractional solution to the LP relaxation of the compact-exponential

ILP, we round the fractional solution to an integer solution by interpreting the fractional values as probabilities of schedules. The obstacle is that the compact-exponential LP relaxation is exponential in size. We resort to its dual that has a polynomial number of variables and an exponential number of constraints. We then employ the ellipsoid algorithm [22] and design a new separation oracle to separate violated constraints. The primal variables corresponding to the violated dual constraints can be selected. Consequently, we derive a new polynomial-sized LP from the original compact-exponential LP, which can be solved in polynomial time. We show that the obtained integer solution guarantees an expected $(1-\varepsilon_1)$-optimal objective value, where ε_1 can be arbitrarily close to 0.

We proceed to consider the practical online scheduling version of the problem with stochastic input, and determine the schedule upon the arrival of each job without future information. We apply the primal-dual framework of algorithm design for such online decision making, with dual variables indicating resource prices. To address the exponential size of the compact-exponential LP, we first convert the optimization problem in the online stochastic model into a deterministic fractional program, exploiting the job arrival process. This new program removes the time domain and has a polynomial number of variables. It serves as an upper-bound of the optimal objective value in expectation, and its dual variables act as a threshold for job admission. To approximately obtain a dual solution close to the offline dual optimum, we gradually learn it based on past jobs, and refine it as more jobs arrive. Our online scheduling framework guarantees computational efficiency, and produces a $(1-O(\varepsilon_2))$-optimal objective value in expectation, where ε_2 can be arbitrarily close to 0.

In the rest of the chapter, we discuss related work in Ch. 4.2. We introduce the system model and formulate the optimization problem in Ch. 4.3. Ch. 4.4 and Ch. 4.5 present the offline and online scheduling frameworks, respectively, which are evaluated in Ch. 4.6. Ch. 4.7 concludes the chapter.

4.2 Related Work

Recent literature on cloud computing witnessed a plethora of studies on dynamic VM provisioning, in both offline and online settings [78] [62] [80]. Zhang *et al.* [78] apply a convex decomposition technique to design a randomized algorithm for dynamic cloud resource provisioning, achieving a small approximation ratio. Shi *et al.* [62] further extend the study to an online scenario, where each cloud user is subject to its budget constraint. Zhang *et al.* [62] propose an online algorithm for the stochastic job arrival model. They aim to optimize the packing of VMs to satisfy each job's demand in a fixed time window. The above studies do not consider the scheduling dimension in their solution space. Furthermore, they focus on the allocation of VMs, while we describe a richer model where each job runs over containers characterized by a dependence graph.

Towards job scheduling under the full execution mode, Baruah *et al.* [19] study the traditional all-or-no-value model, and prove a tight bound on the competitive ratio for the online scheduling problem. Koren *et al.* [48] propose D-over, an algorithm that achieves the same competitive ratio. The above literature considers only one type of resource. In Chapter 3 we presented online scheduling algorithms for cloud computing jobs with soft deadlines. The solution there relies on information about the minimum and the maximum unit values of resources, which are sometimes hard to obtain in the online setting.

Earlier studies on partial job execution often assume no resource sharing and focus on preemptive scheduling [29] [27]. Recent studies investigate cloud jobs with partial values. Navendu *et al.* [44] design two scheduling mechanisms for computing jobs with deadlines in the offline scenario. They consider only one type of resource, and guarantee an approximation ratio that is relatively loose. Lucier *et al.* [53] propose online scheduling algorithms for deadline-sensitive jobs in a simple model, where each job contains a single subtask. Azar *et al.* [18] further improve the algorithm and analyze its competitive ratio. Both studies assume that a server can only execute one job at each time slot. Zheng *et al.* [82] design online multi-resource allocation algorithms that allow partial execution of jobs, achieving small competitive ratios. Their model assumes that all

subtasks of a job are identical and have no inter-dependence. We aim to design general scheduling frameworks for cloud container services, targeting small approximation ratio and competitive ratio in the offline and online settings, respectively.

Our offline algorithm combines the ellipsoid algorithm [22] with the randomized rounding technique [59], which is partially inspired by Fleischer *et al.* [33]. However, they focus on rather different problems – maximum general assignment problems. For theoretical research on online stochastic algorithm design, Agrawal *et al.* [8] study a general online packing problem, and propose a simpler and fast primal-dual algorithm for it. They rely on a one-time learning process while our work performs a dynamic learning process. Kesselheim *et al.* [46] study online packing LPs in the random order model. They solve an LP in every step, and round the fractional solution to an integer solution. Gupta *et al.* [40] consider the problem of solving packing/covering LPs online, and construct primal solutions based on dual solutions through a regret-minimizing online learning algorithm. Jaillet *et al.* [43] study the online dynamic resource allocation problem, and propose a learning-based algorithm. Agrawal *et al.* [9] apply a similar idea to the general online optimization problem. Different from the above literature [8] [46] [40] [43] [9], we do not require the number of inputs to be known in advance. Furthermore, prior work considers a more general form of the problem but limits the number of schedules for each job to a small number. Such techniques can suffer from exponential blowup in problem size when considering jobs with subtasks, since each job has an exponential number of possible schedules. In this chapter, we focus on a particular form of packing problem that formulates the scheduling problem for cloud container services, develop methods that are more computationally tractable and better tailored to those settings, and then evaluate those methods empirically.

4.3 System Model

We consider a cloud service provider, which hosts a pool of K types of resources, as exemplified by CPU, RAM and disk storage. Cloud resources can be dynamically packed into different containers

on demand. Let $[X]$ denote the integer set $\{1,2,\ldots,X\}$. For each type-k ($k \in [K]$) resource, there is a total of c_k units in the cloud.

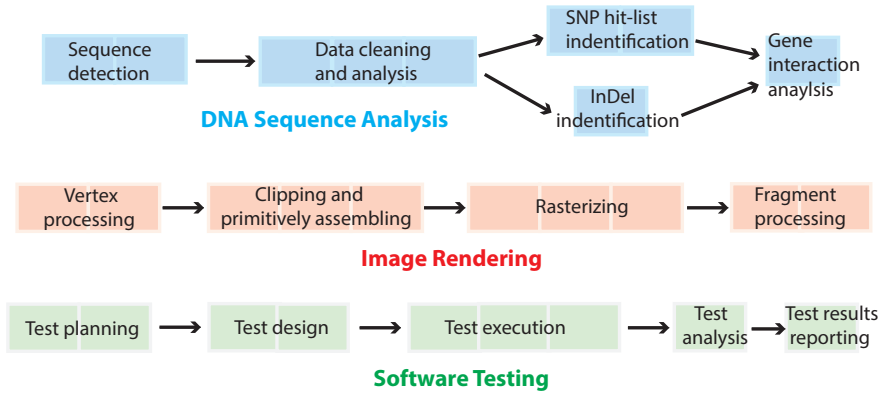

Figure 4.1: Dependence graphs for cloud computing jobs.

Assume the job arrival process during a large time span $[T] = 1,2,\ldots,T$ is a Poisson process with rate λ. Recall that a Poisson process has the following properties [69]: i) the total number of job arrivals in T time slots, I, is a random variable following the Poisson distribution with an expected value of λT; ii) the arrival time of each job can be uniformly and independently mapped to a slot in $[T]$. Our online algorithm design and analysis are based on this assumption. However, we do not require that the job arrival process must follow a Poisson process. Our online algorithm can work on more general arrival processes, as long as the expectation of I can be estimated and property ii) holds. Based on ii), we assume that the arrival time of each job is uniformly and independently drawn from $[T]$, and index jobs according to their order of arrival in any fixed realization of the arrival process. Let $[I] = \{1,2,\ldots,I\}$ be the set of jobs. Each job i consists of multiple subtasks, and is expressed by a tuple

$$\Gamma_i = \{a_i, d_i, N_i, G_i, \{L_{in}\}_{n\in[N_i]}, \{R_{in}\}_{n\in[N_i]}, \{b_{in}\}_{n\in[N_i]}\},$$

where a_i and d_i are the arrival time and the deadline of job i. N_i is the number of subtasks in job i. G_i is the dependence graph that captures the dependencies among subtasks in job i. Example dependence graphs are illustrated in Fig 4.1. The execution of job i's nth subtask doesn't need to be continuous; we only require that the total execution time accumulates to L_{in}. $R_{in} = \{r_{in}^k\}_{k\in[K]}$

is the resource profile of the container that serves job i's nth subtask, where r_{in}^k is the amount of type-k resource required. If job i's nth subtask is completed by d_i, a *partial value* b_{in} is obtained. Let $r_{max}^k = \max_{i\in[I],n\in[N_i]}\{r_{in}^k\}$ denote the maximum type-k resource demand. We refer to $C = \min_{k\in[K]}\{\frac{c_k}{r_{max}^k}\}$ as the *capacity ratio*. Let $N = \max_{i\in[I]}\{N_i\}$, $D = \max_{i\in[I]}\{d_i - a_i\}$ and $L_{\max} = \max_{i\in[I],n\in[N_i]}\{L_{in}\}$. Table 4.1 summarizes notation for easy reference. Each job $i \in [I]$ is drawn independently from a set of job types, $\mathscr{D}$, following an unknown distribution, *i.e.,* job types are *i.i.d.* A job type defines the configuration of a job, including the profiles of its subtasks, *i.e.,* $\{N_i, G_i, \{L_{in}\}_{n\in[N_i]}, \{R_{in}\}_{n\in[N_i]}, \{b_{in}\}_{n\in[N_i]}\}$, and the duration of the job, *i.e,.* $d_i - a_i$. Note that a job's arrival time a_i and deadline d_i are not part of the job type. For example, an access service chain job is configured by "Firewall→IDS→Proxy" with $d_i - a_i = 20$, where instances of firewall, IDS and proxy are encapsulated into containers with predefined resource demands, and it must be deployed within 20 time slots following its arrival.

In practice, there are jobs that render an atomic value B_i only upon completion of all its subtasks before the deadline. This type of job can be viewed as a special case of our model, by setting $b_{i1} = b_{i2} = \cdots = b_{iN_i-1} = 0$ and $b_{iN_i} = B_i$.

Our objective is to maximize the total valuation obtained from all jobs, subject to resource capacity and job scheduling constraints. A binary number $x_{in} \in \{0,1\}$ indicates whether job i's nth subtask is completed (1) or not (0). Let another binary number $y_{in}(t)$ encode the scheduling of job i's nth subtask, where $y_{in}(t) = 1$ if job i's nth subtask is executed at time slot t, and 0 otherwise. Under a fixed realization of the job arrival process, the offline optimization problem can be formulated into the following integer linear program (ILP):

$$\text{maximize} \sum_{i\in[I]}\sum_{n\in[N_i]} b_{in}x_{in} \tag{4.1}$$

subject to:

$$\sum_{t=a_i}^{d_i} y_{in}(t) \geq L_{in}x_{in}, \forall i \in [I], \forall n \in [N_i], \tag{4.1a}$$

$$ty_{in}(t) < t'y_{in'}(t'), \forall t : y_{in}(t) = 1, \forall t' : y_{in'}(t') = 1,$$

$$\forall i : n \text{ is } n'\text{'s ancestor}, \tag{4.1b}$$

$$\sum_{i\in[I]}\sum_{n\in[N_i]} r^k_{in} y_{in}(t) \le c_k, \forall k \in [K], \forall t \in [T], \tag{4.1c}$$

$$x_{in}, y_{in}(t) \in \{0,1\}, \forall i \in [I], \forall n \in [N_i], \forall t \in [a_i, d_i]. \tag{4.1d}$$

Constraint (4.1a) guarantees that the number of allocated time slots between job i's arrival time and deadline is sufficient to serve its nth subtask. Constraint (4.1b) enforces the execution sequence of job i's subtasks based on its dependence graph. The capacity of type-k resource is formulated in constraint (4.1c).

Even in the offline setting, with complete knowledge of the system given, the polynomial-sized ILP (4.1) is still NP-hard to solve. The classic multidimensional knapsack problem, which is known to be NP-hard, can be reduced to a special case of ILP (4.1) by setting $T = 1 = N_i = L_{in} = 1$. The challenge further escalates when we involve unconventional job scheduling constraints (constraints (4.1a) and (4.1b)). To address these challenges, we first apply the *compact-exponential* technique [84] to reformulate ILP (2) into an equivalent conventional ILP with packing structure, at the cost of introducing an exponential number of variables:

$$P: \quad \text{maximize} \quad \sum_{i\in[I]}\sum_{l\in\zeta_i} b_{il} x_{il} \tag{4.2}$$

subject to:

$$\sum_{i\in[I]}\sum_{l:t\in T(l)} f^k_{il}(t) x_{il} \le c_k, \forall k \in [K], \forall t \in [T], \tag{4.2a}$$

$$\sum_{l\in\zeta_i} x_{il} \le 1, \forall i \in [I], \tag{4.2b}$$

$$x_{il} \in \{0,1\}, \forall i \in [I], \forall l \in \zeta_i. \tag{4.2c}$$

In the above *compact-exponential* ILP, ζ_i is the set of feasible time schedules for job i. A feasible time schedule is a vector $l = \{y_{in}(t)\}$ that satisfies constraints (4.1a) and (4.1b). Variable $x_{il} \in \{0,1\}$ indicates whether job i's schedule l is accepted (1) or not (0). b_{il} is the value based on

the number of completed subtasks. $T(l)$ records the set of time slots in l. $f_{il}^{k}(t)$ denotes the total type-k resource occupation of job i's schedule l in t. Constraints (4.2a) are equivalent to (4.1c). Constraints (4.2b) ensure that each job is executed according to at most one schedule.

We relax $x_{il} \in \{0,1\}$ to $x_{il} \geq 0$, and introduce dual variables $p_k(t)$ and u_i for constraints (4.2a) and (4.2b). The dual of the relaxed problem (4.2) is:

$$D: \quad \text{minimize} \sum_{t\in[T]} \sum_{k\in[K]} c_k p_k(t) + \sum_{i\in[I]} u_i \tag{4.3}$$

subject to:

$$u_i \geq b_{il} - \sum_{k\in[K]} \sum_{t\in T(l)} f_{il}^{k}(t) p_k(t), \forall i \in [I], \forall l \in \zeta_i, \tag{4.3a}$$

$$p_k(t), u_i \geq 0, \forall i \in [I], \forall k \in [K], \forall t \in [T]. \tag{4.3b}$$

It is clear that a feasible solution to ILP (4.2) has a corresponding feasible solution in ILP (4.1), and the two ILPs have the same optimal objective value. Our offline algorithm design doesn't rely on any assumption about the job arrival process and job types, while our online algorithm design resorts to the help of them and considers the expected version of the original problem. We first focus on the offline scenario where all jobs are known in advance.

Table 4.1: Summary of Notation in Chapter 4

I	# of jobs	S	$\log_2(\frac{1}{\varepsilon_2}) - 1$	T	# of time slots	r_{max}^{k}	$\max_{i\in[I],n\in[N_i]}\{r_{in}^{k}\}$
λ	job arrival rate	a_i	job i's arrival time	$\mathscr{D}$	job types set	d_i	job i's deadline
C	$\min_{k\in[K]}\{\frac{c_k}{r_{max}^{k}}\}$	$L_{\max}$	$\max_{i\in[I],n\in[N_i]}\{L_{in}\}$	N	$\max_{i\in[I]}\{N_i\}$	D	$\max_{i\in[I]}\{d_i - a_i\}$

$[X]$	integer set $\{1,\ldots,X\}$	N_i	# number of subtasks/containers of job i
K	# of types of resources	L_{in}	# of time slots requested by job i's nth subtask
c_k	capacity of type-k resource	r_{in}^{k}	demand of type-k resource by job i's nth subtask
G_i	job i's dependence graph	x_{in}	job i's nth subtask is completed (1) or not (0)
b_{in}	value of job i's nth subtask	$y_{in}(t)$	whether or not to allocate job i's nth subtask in t
$f_{il}^{k}(t)$	type-k resource occupation of job i's schedule l in t		

4.4 Offline Scheduling Framework

In this section, we design a randomized scheduling algorithm for the offline setting, when future job information is available or can be predicted. We first solve the LP relaxation of compact-exponential ILP (4.2) approximately in Ch. 4.4.1, and then round the fractional solution to a feasible integer solution of ILP (4.1) in Ch. 4.4.2.

4.4.1 Solving the Compact-Exponential ILP

ILP (4.2) has an exponential number of variables, each corresponding to a possible schedule for job i. To solve ILP (4.2), we first solve its dual problem (4.3), which has a polynomial number of variables but an exponential number of constraints. We rewrite LP (4.3) to the following covering problem:

$$\text{minimize} \sum_{t\in[T]} \sum_{k\in[K]} c_k p_k(t) + \sum_{i\in[I]} u_i \tag{4.4}$$

subject to:

$$(u_i, \{p_k(t)\}_{k\in[K],t\in[T]}) \in P_i, \forall i \in [I], \tag{4.4a}$$

$$p_k(t), u_i \geq 0, \forall i \in [I], \forall k \in [K], \forall t \in [T]. \tag{4.4b}$$

Here P_i is the polytope for job i defined by constraints of the form $u_i \geq b_{il} - \sum_{k\in[K]} \sum_{t\in T(l)} f_{il}^k(t) p_k(t)$, $\forall l \in \zeta_i$. We resort to a separation oracle for P_i, *i.e.,* an algorithm that, given an input of dual variables $(u_i, \{p_k(t)\}_{k\in[K],t\in[T]})$, returns either a violated constraint, or guarantees that $(u_i, \{p_k(t)\}_{k\in[K],t\in[T]})$ is feasible for P_i.

If we interpret $p_k(t)$ as the marginal price of type-k resource at time t, then $b_{il} - \sum_{k\in[K]} \sum_{t\in T(l)} f_{il}^k(t) p_k(t)$ is the utility of job i executed by schedule l. We can use a scheduling algorithm for the utility maximization problem for job i to design a separation oracle for P_i, as follows. Given the marginal price $\{p_k(t)\}_{k\in[K],t\in[T]}$, utility maximization for job i requires finding a schedule l^* with value u_i^* such that for any schedule $l \in \zeta_i$, $u_i^* = b_{il^*} - \sum_{k\in[K]} \sum_{t\in T(l^*)} f_{il^*}^k(t) p_k(t) \geq b_{il} - \sum_{k\in[K]} \sum_{t\in T(l)} f_{il}^k(t) p_k(t)$. Then either $u_i < u_i^*$ or $u_i \geq u_i^*$. If $u_i < u_i^*$, then a violated constraint

with schedule l^* is found. Otherwise, $u_i \geq u_i^* \geq b_{il} - \sum_{k\in[K]}\sum_{t\in T(l)} f_{il}^k(t)p_k(t)$ for any $l \in \zeta_i$ and $(u_i, \{p_k(t)\}_{k\in[K],t\in[T]})$ is feasible for P_i.

Algorithm 5 A Separation Oracle for Polytope P_i - Chain Structure

Input: $(u_i, \{p_k(t)\}_{k\in[K],t\in[T]}), \Gamma_i$

1: Calculate $c_n(t) = \sum_{k\in[K]} r_{in}^k p_k(t), \forall n \in [N_i], t \in [a_i, d_i]$;
2: **for** $\eta = 1, \ldots, N_i$ **do**
3: **for** $n \in [\eta]$ **do**
4: **for** $t_s \in [a_i + \sum_1^{n-1} L_{in}, d_i - \sum_n^{\eta} L_{in} + 1]$ **do**
5: **for** $t_e \in [t_s + L_{in} - 1, d_i - \sum_{n+1}^{\eta} L_{in}]$ **do**
6: Select L_{in} slots between t_s and t_e with minimum $c_n(t)$, and save them to $\tau_n(t_s, t_e)$;
7: $P_n(t_s, t_e) = \sum_{t\in\tau_n(t_s,t_e)} c_n(t)$;
8: **end for**
9: **end for**
10: **if** $n > 1$ **then**
11: **for** $t_s \in [a_i + \sum_1^{n-1} L_{in}, d_i - \sum_n^{\eta} L_{in} + 1]$ **do**
12: $t_s^*, t_e^* = \arg\min_{t_e' < t_s} \{P_{n-1}(:, t_e')\}$;
13: $P_n(t_s, t_e) = P_n(t_s, t_e) + P_{n-1}(t_s^*, t_e^*), \forall t_e$;
14: $\tau_n(t_s, t_e) = \tau_{n-1}(t_s^*, t_e^*) \cup \tau_n(t_s, t_e), \forall t_e$;
15: **end for**
16: **end if**
17: **end for**
18: $t_s^\eta, t_e^\eta = \arg\min_{t_s,t_e} \{P_\eta(t_s, t_e)\}$;
19: $l_\eta = \tau_\eta(t_s^\eta, t_e^\eta)$; $U_\eta = \sum_{n=1}^{\eta} b_{in} - P_\eta(t_s^\eta, t_e^\eta)$;
20: **end for**
21: $\eta^* = \arg\max_\eta \{U_\eta\}, l^* = l_{\eta^*}$;
22: **if** $U_{\eta^*} \geq 0$ **then**
23: **Output:** $(u_i, \{p_k(t)\}_{k\in[K],t\in[T]}) \in P_i$;
24: **else** **Output:** A violated constraint with l^*.
25: **end if**

We focus on a special type of job with a sequential chain structure, which are often adopted by service chains in the recent paradigm of NFV [39]. Generalization to jobs with general directed acyclic graphs is left as future work. Algorithm 5 is a separation oracle for P_i, which exactly solves the utility maximization problem for job i. The construction of the best schedule that maximizes job i's utility is based on a dynamic programming approach. We first calculate the price of container n running at time t in line 1. Because job i consists of N_i subtasks each with a partial value b_{in}, we use a `for` loop (lines 2-20) to compute the best schedule l_η if η subtasks are completed before

the deadline. For the nth subtask, we calculate the cheapest schedule $\tau_n(t_s, t_e)$ to finish it within a given time period $[t_s, t_e]$ and its corresponding price in lines 4-9. Because the $(n-1)$th subtask must be completed before nth subtask ($n > 1$), we also fix the schedule of the $(n-1)$th subtask when considering nth subtask's schedule, by choosing the cheapest schedule that completes the $(n-1)$th subtask before t_s in lines 10-16. Lines 18-19 compute the best schedule and job i's utility if η subtasks are completed. Lines 21-25 figure out job i's final utility and output the result.

Lemma 11. The time complexity of the separation oracle in Algorithm 5 is polynomial.

Proof: Line 1 takes $O(KN_i(d_i - a_i))$ steps to calculate the price in each time slot. The first `for` loop iterates N_i times and the second `for` loop iterates at most N_i times. Within the second `for` loop, lines 4-9 include two `for` loops to select the best schedule within a given time period and compute its price, which can be done in $O((d_i - a_i)^3 L_{in})$ steps, since the execution time in line 6 is $O((d_i - a_i)L_{in})$. Lines 10-16 update the schedule and its price, taking $O((d_i - a_i)^3)$ steps. Therefore, the execution time for the second `for` loop (lines 3-17) is $O(N_i(d_i - a_i)^3 L_{\max})$ with $L_{\max} = \max_{i\in[I], n\in[N_i]}\{L_{in}\}$. Lines 18-19 take $O((d_i - a_i)^2)$ steps to compute the best schedule. Hence, the running time from line 2 to 20 is $O(N_i^2(d_i - a_i)^3 L_{\max})$. The `if` statement in lines 21-25 returns the final output within $O(N_i)$ steps. In summary, the overall running time of Algorithm 5 is $O(KN_i^2(d_i - a_i)^3 L_{\max})$. □

Lemma 12. For any $0 < \alpha < 1$, given a polynomial-time separation oracle for P_i, we can design a $\frac{1}{1-\alpha}$-approximation algorithm to solve the LP (4.3) and hence the LP relaxation of ILP (4.2) in polynomial time.

Proof: We run the ellipsoid method on LP (4.3), using Algorithm 5 as a separation oracle. More precisely, we start with an estimate of the maximum objective value of LP (4.3), v_0 (*e.g.,* $v_0 = \sum_{i\in[I]}\sum_{n\in[N_i]} b_{in}$), and use the ellipsoid algorithm to check the feasibility of the following linear

constraints:

$$\sum_{t\in[T]}\sum_{k\in[K]} c_k p_k(t) + \sum_{i\in[I]} u_i \le v_0,$$

$$u_i \ge b_{il} - \sum_{k\in[K]}\sum_{t\in T(l)} f_{il}^k(t) p_k(t),\ \forall i \in [I], \forall l \in \zeta_i,$$

$$p_k(t), u_i \ge 0, \forall i \in [I],\ \forall k \in [K], \forall t \in [T].$$

If this LP is feasible, we know that the optimal objective value of LP (4.3) is at most v_0. We now decrease v_0 to $v_0/2$, and check the feasibility again. If this is true, we know the optimum lies in $(0, v_0/2]$. This is essentially a binary search to find the smallest feasible objective value. Let D^* denote the optimal objective value of LP (4.3). Suppose $v_0 \le h \cdot D^*$, then after $\lfloor \log_2 h \rfloor + \lceil \log_2 \frac{1}{\alpha} \rceil$ steps, we terminate at an interval $(v^* - \alpha v^*, v^*]$, with a solution $(\{u_i\}, \{p_k(t)\})$ such that $v^* = \sum_{t\in[T]}\sum_{k\in[K]} c_k p_k(t) + \sum_{i\in[I]} u_i$. Let D be the current dual objective value and $D = v^*$. Furthermore, we have $v^* - \alpha v^* \le D^* \le v^*$. To check the feasibility of one point, the ellipsoid method calls the separation oracle $O(I^3 \mathscr{L})$ times where each job is encoded in $\mathscr{L}$ bits [22]. Thus, we obtain a solution to LP (4.3) after $O(I^3(\log h + \log \frac{1}{\alpha}))$ iterations of the separation oracle. Because the running time of the separation oracle in Algorithm 5 is polynomial, the overall running time to solve LP (4.3) is also polynomial.

In the execution of the ellipsoid algorithm to check the feasibility of $v^* - \alpha v^*$, only a polynomial number of dual constraints (4.3a) are involved. This set of constraints is sufficient to show the objective value of LP (4.3) is greater than $v^* - \alpha v^*$. To solve the LP relaxation of ILP (4.2), we only need to consider a polynomial number of variables corresponding to this set of dual constraints (by setting all other variables to zero). Thus, this polynomial-sized LP can be solved in polynomial time (*e.g.,* using Karmarkar's algorithm [68], its running time is $O(I^{3.5})\mathscr{L}$). Let P be the objective value, with $P > v^* - \alpha v^*$ by LP duality. Let P^* be the optimal objective value of the relaxed LP (4.2). By LP duality,

$$\frac{P}{P^*} \ge \frac{P}{D} > \frac{v^* - \alpha v^*}{v^*} = (1-\alpha),$$

we obtain a $\frac{1}{1-\alpha}$-approximation algorithm. The running time of this algorithm is polynomial, which is $O(I^{3.5}(\log h + \log \frac{1}{\alpha})KN^2D^3L_{\max}\mathscr{L})$ with $N = \max_{i\in[I]}\{N_i\}$ and $D = \max_{i\in[I]}\{d_i - a_i\}$.

□

4.4.2 A Randomized Offline Scheduling Algorithm

Given a fractional solution to ILP (4.2), we continue to design a near-optimal offline algorithm to schedule jobs based on the randomized rounding technique [59]. $A_{offline}$ in Algorithm 6 is our offline scheduling algorithm. We first solve the LP relaxation of ILP (4.2) in line 1 using the ellipsoid method introduced in the previous subsection. Then we round the fractional solution x^f_{il} to an integer solution in lines 2-5. In order to increase the feasibility of the integer solution, we generate a solution according to a scaled probability, *i.e.,* we select each schedule l with probability $(1-\frac{\varepsilon'}{2})x^f_{il}$ for job i, where $0 < \varepsilon' < 1$, and reject job i with probability $1 - \sum_{l\in\zeta_i}(1-\frac{\varepsilon'}{2})x^f_{il}$. We will show that with high probability (see Theorem 7), our integer solution is feasible. We first bound the probability that one of constraints (4.2a) is violated during the rounding of the fractional solution.

Theorem 6. Chernoff Bound [15] [59]. Let $X_1,\ldots,X_N$ be independent Poisson trials such that, for $1 \le n \le N$, $Pr[X_n = 1] = p_n$, where $0 \le p_n \le 1$. Then for $X = \sum_{n=1}^{N} X_n, \mu \ge E[X] = \sum_{n=1}^{N} p_n$ and $0 < \delta < 2e-1$, we have

$$Pr[X > (1+\delta)\mu] < e^{-\mu\delta^2/4}.$$

Lemma 13. In our cloud system, assume the capacity ratio $C \ge \frac{16(c+1)}{\varepsilon'^2}\ln(KT)$ with $c > 0$. Let Φ denote the event that the amount of allocated type-k resource at time t exceeds c_k, then the probability that event Φ happens is at most $\frac{1}{(KT)^{c+1}}$.

Proof: Recall that C is defined as $\min_{k\in[K]}\{\frac{c_k}{r^k_{max}}\}$ and Φ is the event that constraint (4.2a) is violated.

Algorithm 6 A Randomized offline Algorithm $A_{offline}$

Input: $\{\Gamma_i\}_{i\in[I]}, \{c_k\}_{k\in[K]}$, $0 < \varepsilon' < 1$

1: Solve the LP relaxation of ILP (4.2) using the ellipsoid method. Let the solution be $\{x^f_{il}\}_{i\in[I],l\in\zeta_i}$;
2: **for** each job i **do**
3: Choose each schedule l with probability $(1-\frac{\varepsilon'}{2})x^f_{il}$, and reject job i with probability $1-\sum_{l\in\zeta_i}(1-\frac{\varepsilon'}{2})x^f_{il}$;
4: If schedule l^* is selected, set $x_{il^*}=1$; Update the corresponding $\{x_{in}\}_{n\in[N_i]}$ and $\{y_{in}(t)\}_{n\in[N_i],t\in[T]}$ according to schedule l^*;
5: Schedule job i accord to $y_{in}(t)$;
6: **end for**

For given k and t, we have

$$Pr[\Phi] = Pr[\sum_{i\in[I]}\sum_{l:t\in T(l)} f^k_{il}(t)x_{il} > c_k] \le Pr[\sum_{i\in[I]}\sum_{l:t\in T(l)} r^k_{max}x_{il} > c_k]$$
$$= Pr[\sum_{i\in[I]}\sum_{l:t\in T(l)} x_{il} > \frac{c_k}{r^k_{max}}] \le Pr[X > C],$$

where $X = \sum_{i\in[I]}\sum_{l:t\in T(l)} x_{il}$. Instead of constraints (4.2a), we consider the following LP with new constraints $\sum_{i\in[I]}\sum_{l\in\zeta_i} x_{il} \le C$:

$$\text{maximize} \quad \sum_{i\in[I]}\sum_{l\in\zeta_i} b_{il}x_{il} \tag{4.5}$$

subject to:

$$\sum_{i\in[I]}\sum_{l\in\zeta_i} x_{il} \le C, \forall k\in[K], \forall t\in[T], \tag{4.5a}$$

$$\sum_{l\in\zeta_i} x_{il} \le 1, \forall i\in[I], \tag{4.5b}$$

$$x_{il} \ge 0, \forall i\in[I], \forall l\in\zeta_i. \tag{4.5c}$$

Let $\hat{x}^f$ be the solution to LP (4.5) obtained by the ellipsoid method, and x' be an integer solution to LP (4.5) computed by the same method in lines 3-4 in Algorithm 6. Then $Pr[x'_{il}=1] = (1-\frac{\varepsilon'}{2})\hat{x}^f_{il}$. Let $X'_i = \sum_{l\in\zeta_i} x'_{il}$ and $X' = \sum_{i\in[I]} X'_i$. By the union bound,

$$Pr[X'_i = 1] \le \sum_{l\in\zeta_i} Pr[x'_{il}=1] = (1-\frac{\varepsilon'}{2})\sum_{l\in\zeta_i}\hat{x}^f_{il}.$$

Hence,

$$E[X'] = \sum_{i\in[I]} Pr[X'_i = 1] \leq (1-\frac{\varepsilon'}{2})C.$$

Let $\mu = (1-\frac{\varepsilon'}{2})C$ and $\delta = \frac{\frac{\varepsilon'}{2}}{1-\frac{\varepsilon'}{2}}$. Because $C \geq \frac{16(c+1)}{\varepsilon'^2}\ln(KT)$, $\mu \geq E[X']$ and $0 < \delta < 2e-1$, the following inequality holds by applying the Chernoff bound in Theorem 6:

$$\begin{aligned} Pr[X' > C] &< \exp\left(-(1-\frac{\varepsilon'}{2})C(\frac{\frac{\varepsilon'}{2}}{1-\frac{\varepsilon'}{2}})^2/4\right) \\ &\leq \exp\left(-\frac{c+1}{1-\frac{\varepsilon'}{2}}\ln(KT)\right) = (KT)^{-\frac{c+1}{1-\frac{\varepsilon'}{2}}} \leq \frac{1}{(KT)^{c+1}}. \end{aligned}$$

Therefore, we obtain

$$Pr[\Phi] \leq Pr[X > C] \leq Pr[X' > C] \leq \frac{1}{(KT)^{c+1}}.$$

□

Theorem 7. If $C \geq \frac{16(c+1)}{\varepsilon'^2}\ln(KT)$, with probability at least $1-\frac{1}{(KT)^c}$, $A_{offline}$ in Algorithm 6 can output a feasible solution to ILP (4.1) and ILP (4.2) in polynomial running time. The expected value returned by it is at least $(1-\varepsilon_1)$-optimal, where $\varepsilon_1 = \alpha + \frac{\varepsilon'}{2} - \frac{\alpha\varepsilon'}{2} + \frac{1}{(KT)^c} - (\alpha + \frac{\varepsilon'}{2} - \frac{\alpha\varepsilon'}{2})\frac{1}{(KT)^c}$.

Proof: We first examine the feasibility and the running time. Taking a union bound on K types of resources and T time slots, the probability that the integer solution generated at line 4 in Algorithm 6 is feasible is at least $1-KT\frac{1}{(KT)^{c+1}} = 1-\frac{1}{(KT)^c}$ by Lemma 13. By Lemma 12, line 1 in Algorithm 6 takes polynomial time to compute a fractional solution. The running time of the `for` loop in lines 2-5 is linear. Thus, the running time of Algorithm 6 is polynomial.

Let AS denote the event that $A_{offline}$ outputs a feasible solution. Let OPT^f be the optimal objective value of the relaxed problem of (4.2), the expected objective value returned by Algorithm

6 is:

$$
\begin{aligned}
&E[\sum_{i\in[I]}\sum_{l\in\zeta_i} b_{il}x_{il}] \geq E[\sum_{i\in[I]}\sum_{l\in\zeta_i} b_{il}x_{il}|AS] \\
&\geq \sum_{i\in[I]}\sum_{l\in\zeta_i} b_{il}E[x_{il}]Pr[AS] \geq \sum_{i\in[I]}\sum_{l\in\zeta_i} b_{il}(1-\frac{\varepsilon'}{2})x_{il}^f\cdot(1-\frac{1}{(KT)^c}) \\
&\geq (1-\frac{\varepsilon'}{2})(1-\alpha)(1-\frac{1}{(KT)^c})OPT^f = (1-\varepsilon_1)OPT^f.
\end{aligned}
$$

Because the optimal objective value of ILP (4.2) is at most OPT^f, we can conclude that Algorithm 6 returns a $(1-\varepsilon_1)$-optimal solution in expectation with $\varepsilon_1 = \alpha + \frac{\varepsilon'}{2} - \frac{\alpha\varepsilon'}{2} + \frac{1}{(KT)^c} - (\alpha + \frac{\varepsilon'}{2} - \frac{\alpha\varepsilon'}{2})\frac{1}{(KT)^c}$. □

4.5 Online Scheduling Framework

A practical scheduling algorithm needs to work in the online fashion, without relying on knowledge of future job arrivals. In this section, we design an online algorithm that runs as jobs arrive to the system, and processes each job immediately upon its arrival. We next introduce the primal-dual framework that guides our online algorithm design in Ch. 4.5.1. We propose an online algorithm for jobs with chain structure in Ch. 4.5.2 and analyze its performance in Ch. 4.5.3. Ch. 4.5.4 shows that the algorithm proposed in Ch. 4.5.2 can also handle general jobs with directed acyclic graph structures.

4.5.1 Primal and Dual Framework

Upon each job arrival, the cloud service provider needs to determine whether to serve this job, and if so, how to schedule it. This process is equivalent to choosing a feasible solution to ILP (4.1). To solve ILP (4.1), we resort to the classic primal-dual framework, and apply it to the compact-exponential ILP (4.2) and its dual (4.3). We observe that for each primal variable x_{il}, there is a dual constraint associated with it. Complementary slackness indicates the update of the primal variable is based on its dual constraint. x_{il} remains zero unless its associated dual constraint (4.2a) is tight.

Let $\mathbf{p}^*$ denote the optimal solution of dual variables $\{p_k(t)^*\}_{\forall k\in[K],t\in[T]}$ for LP (4.3). Upon the arrival of the ith job, we assign dual variable u_i to the maximum of 0 and the right hand side (RHS) of (4.2a),

$$u_i = \max\{0, \max_{l\in\zeta_i}\{b_{il} - \sum_{k\in[K]}\sum_{t\in T(l)} f_{il}^k(t)p_k(t)^*\}\}. \tag{4.6}$$

If $u_i > 0$, the cloud service provider serves job i according to the schedule that maximizes the RHS of constraint (4.2a); If $u_i \leq 0$, the cloud service provider rejects it. The rationale is as follows: The dual variable $p_k(t)^*$ can be interpreted as the marginal price per unit of type-k resource at time t, then $\sum_{k\in[K]}\sum_{t\in T(l)} f_{il}^k(t)p_k(t)^*$ is the price to execute job i according to schedule l. The RHS of (4.2a) can be viewed as job i's utility with schedule l. The assignment of u_i in (4.6) effectively maximizes job i's utility, towards achieving the maximum value obtained from all jobs.

However, the problem is that we cannot obtain the optimal dual solution $\mathbf{p}^*$ in the online setting. We have information on past jobs only. Thus, we consider the first $\varepsilon_2 \in (0,1)$ fraction of jobs and hope to obtain an approximate dual solution in expectation, and progressively refine our dual solution as more jobs arrive. By adopting this idea, we next design an online algorithm, and show that it has good performance in both theoretical analysis and simulation studies.

4.5.2 An Online Algorithm with Stochastic Input

We first focus on service chain type of jobs where the dependence graph is of a sequential chain structure.

Expected offline optimization problem. The offline problem in (4.2) is defined under a fixed and stochastic realization of the job arrival process. Next, we consider all possible realizations of the job arrival process in expectation, and define the expected offline problem in LP (4.7). We refer to it as the *expected offline program*. It guides our online algorithm design and the optimal objective value of it serves as an upper bound of the expected optimal objective value of the offline problem in (4.2) in the competitive ratio analysis.

We use j to denote a job of type-j instead of job j in LPs (4.7), (4.8) and (4.9). Let ρ_j be

the probability that a type-j job is drawn from the job types set $\mathscr{D}$. Since the expected number of jobs is λT, the expected number of type-j jobs appearing in the realized jobs is $\lambda T \rho_j$. Let x_{jl} be the probability of a type-j job served according to schedule l, over a random realization of jobs. Then $\lambda T \rho_j \sum_{l \in \zeta_j} b_{jl} x_{jl}$ is the contribution of type-j jobs to the expected overall obtained value. Summing over all job types, the objective function of (4.7) represents the expected value obtained from all jobs. Note that we assume the same type of jobs has the same value of $d_i - a_i$ regardless of job arrival time, under the assumption that T is much larger than the value of $d_i - a_i$. Because the probability of $d_i > T$ is very small and the overall obtained value in expectation barely changes without considering these extreme jobs.

$$\text{maximize} \quad \sum_{j \in \mathscr{D}} \lambda T \rho_j \sum_{l \in \zeta_j} b_{jl} x_{jl} \tag{4.7}$$

subject to:

$$\sum_{j \in \mathscr{D}} \sum_{l \in \zeta_j} \lambda T \rho_j \frac{\sum_{t \in T(l)} f_{il}^k(t)}{T} x_{jl} \leq c_k, \forall k \in [K], \tag{4.7a}$$

$$\sum_{l \in \zeta_j} x_{jl} \leq 1, \forall j \in \mathscr{D}, \tag{4.7b}$$

$$x_{jl} \geq 0, \forall j \in \mathscr{D}, \forall l \in \zeta_j. \tag{4.7c}$$

Next, we examine the constraints in LP (4.7). Constraint (4.7a) is the expected capacity constraint, which guarantees the average consumption of one type of resource at each slot is below its capacity. The rationale is as follows: If a type-j job is scheduled according to l, then it consumes a total of $\sum_{t \in T(l)} f_{il}^k(t)$ units of type-k resources over the entire system time (T slots). Recall that the arrival time of a job is uniformly distributed within $[T]$, then the slot $t \in T(l)$ is also uniformly distributed within $[T]$. On average over time, a type-j job served with schedule l consumes at most $\frac{\sum_{t \in T(l)} f_{il}^k(t)}{T}$ units of type-k resource at each time slot, since the probability of this job occupying any slot is $1/T$. $\sum_{j \in \mathscr{D}} \sum_{l \in \zeta_j} \lambda T \rho_j \frac{\sum_{t \in T(l)} f_{il}^k(t)}{T} x_{jl}$ is the average consumption of type-k resource at each slot contributed by all types of jobs. Note that it is a non-trivial transformation of the capacity constraints (4.2a) as we remove the time dimension here. Constraint (4.7b) ensures that one job of

a specific type can only be served according to at most one schedule. Based on the above expected offline program, we are able to design an online algorithm that obtains $1 - O(\varepsilon_2)$ fraction of the expected optimal value obtained from all jobs, under the assumption that each job only consumes a small fraction of the capacity of any resource.

While it appears that the number of variables in LP (4.7) is still exponential, we observe that there are only N_j possible values of b_{jl} and $\sum_{t \in T(l)} f_{il}^k(t)$ for each j. This is because a type-j job contains N_j subtasks that need to be executed sequentially, and each of the subtasks has its own value and resource demand. Let $\eta \in [N_j]$ denote the ηth execution option for a type-j job, and $b_{j\eta} = \sum_{n \in [\eta]} b_{jn}$ and $\omega_{j\eta}^k = \sum_{n \in [\eta]} r_{in}^k L_{in}$ represent the value and the resource consumption for this option. We can rewrite LP (4.7) to the following LP:

$$P_\Sigma: \quad \text{maximize} \sum_{j \in \mathscr{D}} \lambda T \rho_j \sum_{\eta \in [N_j]} b_{j\eta} x_{j\eta} \tag{4.8}$$

subject to:

$$\sum_{j \in \mathscr{D}} \sum_{\eta \in [N_j]} \lambda T \rho_j \frac{\omega_{j\eta}^k}{T} x_{j\eta} \leq c_k, \forall k \in [K], \tag{4.8a}$$

$$\sum_{\eta \in [N_j]} x_{j\eta} \leq 1, \forall j \in \mathscr{D}, \tag{4.8b}$$

$$x_{j\eta} \geq 0, \forall j \in \mathscr{D}, \forall \eta \in [N_j]. \tag{4.8c}$$

By introducing dual variables p_k and u_j into constraints (4.8a) and (4.8b), respectively, the dual problem of (4.8) is:

$$D_\Sigma: \quad \text{minimize} \sum_{k \in [K]} c_k p_k + \sum_{j \in \mathscr{D}} \lambda T \rho_j u_j \tag{4.9}$$

subject to:

$$u_j \geq b_{j\eta} - \sum_{k \in [K]} \frac{\omega_{j\eta}^k}{T} p_k, \forall j \in \mathscr{D}, \forall \eta \in [N_j], \tag{4.9a}$$

$$p_k, u_j \geq 0, \forall k \in [K], \forall j \in \mathscr{D}. \tag{4.9b}$$

If we can solve the dual problem in (4.9) exactly to obtain the optimal dual solution $\mathbf{p}_\Sigma$, we can apply the primal-dual technique discussed in Ch. 4.5.1 to derive the primal solution for the expected

offline program (4.8), achieving a close-to-optimal objective value. The barrier is still that we do not have complete knowledge of all job types in the online setting. Our main idea is to produce an approximate dual solution based on past jobs, and gradually refine this dual solution with the accumulation of past jobs. The intuition is that because the types of jobs are *i.i.d.*, the average resource consumption of the past jobs can approximately reflect the average resource consumption of all jobs in expectation, especially when more and more jobs are processed. More specifically, we divide the job arrival process into $\log_2(\frac{1}{\varepsilon_2})$ stages, and index each stage with an integer s. Let $S = \log_2(\frac{1}{\varepsilon_2}) - 1$ and then $s \in [0, 1, \ldots, S]$. For each stage, we consider the first $2^s \lfloor \varepsilon_2 \lambda T \rfloor$ jobs in set $\mathscr{I}_s = [1, \ldots, 2^s \lfloor \varepsilon_2 \lambda T \rfloor]$, and formulate an empirical version of (4.8) in P_s in (4.10) for these sample jobs. We replace the expectations over all jobs in the objective function and constraint (4.8a) with the sum over these jobs, and shrink the capacity limits accordingly by a factor of $(1 - \mathscr{F}_s)2^s\varepsilon_2$. Let $I_s = |\mathscr{I}_s| = 2^s \lfloor \varepsilon_2 \lambda T \rfloor$ and $\mathscr{F}_s = \varepsilon_2 \sqrt{\frac{\lambda T}{2^s \varepsilon_2 \lambda T}} = \sqrt{\frac{\varepsilon_2}{2^s}}$. Then $2^s\varepsilon_2 \approx \frac{I_s}{\lambda T}$ is the proportion of the first $2^s \lfloor \varepsilon_2 \lambda T \rfloor$ jobs to all jobs, and $(1 - \mathscr{F}_s)$ handles the sampling error to make sure the overall resource consumption does not exceed the capacity. Note that $\varepsilon_2 \leq \mathscr{F}_s \leq \sqrt{\varepsilon_2}$, and we convert each job type j back to job i. The dual of the relaxed (4.10) is formulated in (4.11).

$$P_s: \quad \text{maximize} \sum_{i \in \mathscr{I}_s} \sum_{\eta \in [N_i]} b_{i\eta} x_{i\eta} \tag{4.10}$$

subject to:

$$\sum_{i \in \mathscr{I}_s} \sum_{\eta \in [N_i]} \frac{\omega_{i\eta}^k}{T} x_{i\eta} \leq (1 - \mathscr{F}_s) 2^s \varepsilon_2 c_k, \forall k \in [K], \tag{4.10a}$$

$$\sum_{\eta \in [N_i]} x_{i\eta} \leq 1, \forall i \in \mathscr{I}_s, \tag{4.10b}$$

$$x_{i\eta} \in \{0, 1\}, \forall i \in \mathscr{I}_s, \forall \eta \in [N_i]. \tag{4.10c}$$

$$D_s: \quad \text{minimize} \sum_{k \in [K]} (1 - \mathscr{F}_s) 2^s \varepsilon_2 c_k p_k + \sum_{i \in \mathscr{I}_s} u_i \tag{4.11}$$

subject to:

$$u_i \geq b_{i\eta} - \sum_{k\in[K]} \frac{\omega_{i\eta}^k}{T} p_k, \forall i \in \mathscr{I}_s, \forall \eta \in [N_i], \tag{4.11a}$$

$$p_k, u_i \geq 0, \forall k \in [K], \forall i \in \mathscr{I}_s. \tag{4.11b}$$

Upon the arrival of the $2^s\lfloor \varepsilon_2 \lambda T \rfloor$th job, we exactly solve the dual problem in (4.11) to obtain the optimal dual solution $\mathbf{p}_s$. The size of the dual problem (4.11) is polynomial, and hence it can be solved efficiently by Karmarkar's algorithm [68]. By involving more and more jobs in solving (4.11), we progressively learn a dual solution that is close to the optimal dual solution $\mathbf{p}_\Sigma$ of the offline dual problem in (4.9).

We next discuss the decision making and the scheduling process, based on the learned dual solution $\mathbf{p}_s$. Upon the arrival of each job, we let u_i be the maximum of 0 and the RHS of constraints (4.11a), *i.e.,*

$$u_i = \max\{0, \max_{\eta\in[N_i]} \{b_{i\eta} - \sum_{k\in[K]} \frac{\omega_{i\eta}^k}{T} p_{k,s}\}\}.$$

If $u_i \leq 0$, then the cloud service provider rejects this job; If $u_i > 0$, the cloud service provider accepts this job, and serves it according to the following schedule: Let $\eta_i = \arg\max_{\eta\in[N_i]} \{b_{i\eta} - \sum_{k\in[K]} \frac{\omega_{i\eta}^k}{T} p_{k,s}\}$, subtasks $1,\ldots,\eta_i$ in job i will be allocated sequentially to slots from a_i to $a_i + \sum_{n\in[\eta_i]} L_{in} - 1$. Although we didn't check the resource capacity constraints (4.2a) here, we show that with high probability (see Lemma 17), our algorithm satisfies the capacity limit in expectation for any type of resource at any time.

A_{online} in Algorithm 7 is our online algorithm, with the scheduling algorithm A_{core} in Algorithm 8 running for each job. Lines 1-2 in A_{online} define variable I_s and initialize primal and dual variables. Lines 4-5 reject the first $\lfloor \varepsilon_2 \lambda T \rfloor$ jobs as price $\mathbf{p}_0$ is not ready yet. Upon the arrival of the ith job ($i \geq \lfloor \varepsilon_2 \lambda T \rfloor + 1$), lines 6-13 determine whether to serve this job, and if so how to schedule it. More specifically, A_{core} in line 7 is run for job $i \in [2^{s-1}\lfloor \varepsilon_2 \lambda T \rfloor + 1, 2^s \lfloor \varepsilon_2 \lambda T \rfloor]$ with the input $\mathbf{p}_{s-1}$. In A_{core}, lines 1-4 determine the utility variable u_i. If $u_i > 0$, we accept job i, compute its schedule l in line 7 and update all primal variables in lines 6-14. On the arrival of

Algorithm 7 An Online Algorithm A_{online}

Input: $\{\Gamma_i\}, \{c_k\}, \varepsilon_2, \lambda, T$

Define $I_s = 2^s \lfloor \varepsilon_2 \lambda T \rfloor$;
Initialize $s = 0$; Let $x_{in} = 0, y_{in}(t) = 0, x_{il} = 0, u_i = 0, p_k = 0, \forall i \in [I], \forall n \in [N_i], \forall t \in [T], \forall l \in \zeta_i, \forall k \in [K]$ by default;
while the arrival of the ith job **do**
    if $i \leq \lfloor \varepsilon_2 \lambda T \rfloor$ **then**
        Reject job i;
    else
        $\big(\{x_{in}\}, \{y_{in}(t)\}\}\big) = A_{core}\big(\Gamma_i, \{c_k\}, \{p_k\}\}\big)$;
        if $\exists n \in [N_i], x_{in} = 1$ **then**
            Schedule job i according to $y_{in}(t)$;
        else
            Reject job i.
        end if
    end if
    if $i = I_s$ and $s \leq \log_2(\frac{1}{\varepsilon_2}) - 1$ **then**
        Solve the dual LP (4.11) exactly to obtain $\mathbf{p}_s$;
        Let $\{p_k\} = \mathbf{p}_s$; $s = s + 1$;
    end if
end while

Algorithm 8 A Scheduling Algorithm A_{core}

Input: $\Gamma_i, \{c_k\}, \{p_k\}$
Output: $\{x_{in}\}, \{y_{in}(t)\}$

for $\eta = 1, 2, \ldots, N_i$ **do**
    $u_{i\eta} = \sum_{n \in [\eta]} b_{in} - \sum_{k \in [K]} \frac{\sum_{n \in [\eta]} r_{in}^k L_{in}}{T} p_k$;
end for
$u_i = \max\{0, \max_{\eta \in [N_i]}\{u_{i\eta}\}\}$;
if $u_i > 0$ **then**
    $\eta_i = \arg\max_{\eta \in [N_i]}\{u_{i\eta}\}$;
    $l_i = \{a_i, \ldots, a_i + \sum_{n \in [\eta_i]} L_{in} - 1\}; x_{il_i} = 1$;
    $x_{in} = 1, \forall n \in [\eta_i]$; $t = t_i$;
    for $n = 1, \ldots, \eta_i$ **do**
        $index = 1$;
        while $index \leq L_{in}$ **do**
            $y_{in}(t) = 1; t = t + 1; index = index + 1$;
        end while
    end for
end if
Return $\{x_{in}\}, \{y_{in}(t)\}$

$2^s\lfloor\varepsilon_2\lambda T\rfloor$th job, line 15 in A_{online} solves the dual LP (4.11) exactly using all jobs from job 1 to job $2^s\lfloor\varepsilon_2\lambda T\rfloor$. Line 16 updates p_k and s. Note that the last time we update price $\mathbf{p}_s$ is the arrival time of job $2^{\log_2(\frac{1}{\varepsilon_2})-1}\lfloor\varepsilon_2\lambda T\rfloor$. This process is repeated until the last job arrives. Note that our algorithm doesn't require any information about the job type distribution. Furthermore, we can use an estimated value of λ instead of the accurate one. We will show that inaccurate estimation has rather mild impact on the performance in the simulations. We next use a simple example to illustrate the process of A_{online}. Suppose the online system spans 32 time slots. Let $\lambda = 0.5$ and $\varepsilon_2 = \frac{1}{4}$. We reject the first 4 jobs, and solve (4.11) with the input of the first 4 jobs to obtain $\mathbf{p}_0$. From job 4 to job 8, we use $\mathbf{p}_0$ as the price to make a decision and solve (4.11) again with the input of the first 8 jobs to obtain $\mathbf{p}_1$. From job 8 to the last job, $\mathbf{p}_1$ serves as the threshold to determine the winner.

4.5.3 Theoretical Analysis

i) Polynomial running time.

Theorem 8. The time complexity of A_{online} in Algorithm 7 is polynomial.

Proof: We first examine the running time of A_{core}. Lines 1-4 take $O(N_iK)$ steps to compute u_i. The running time of the `if` statement in lines 5-15 is $O(\sum_{n\in[\eta_i]} L_{in})$. In summary, the running time of A_{core} is $O(N_iK + \sum_{n\in[\eta_i]} L_{in})$.

We next analyze the running time of A_{online}. Lines 1-2 define and initialize variables in $O(1)$ steps. There are I jobs, and the running time to handle each job (lines 4-13) is dominated by the running time of A_{core}. The body of the `if` statement in lines 15-16 is executed $\lfloor\log_2(\frac{1}{\varepsilon_2})\rfloor$ times, each iteration solves the dual problem in (4.11) in $O(I^{3.5})\mathscr{L}$ steps using Karmarkar's algorithm [68], where each job is encoded in $\mathscr{L}$ bits. Recall that $N = \max_{i\in[I]}\{N_i\}$ and $L_{max} = \max_{i\in[I],n\in[N_i]} L_{in}$. Given the above, the time complexity of our online algorithm A_{online} is $O(\lfloor\log_2(\frac{1}{\varepsilon_2})\rfloor I^{3.5}\mathscr{L} + IN(K+L))$. □

ii) Feasibility of the original problem.

We next show that with high probability, our online algorithm A_{online} can compute a feasible solution to the original problem (4.2). Constraints (4.2b) and (4.2c) are satisfied trivially. When summing over all $s \in [0,\ldots,S]$, Lemma 17 shows that with probability at least $1-2\varepsilon_2$, accepted jobs consume at most the maximum capacity in expectation for any type of resource at any time (*i.e.,* Constraint (4.2a) is satisfied).

Let $x_{i\eta}(\mathbf{p}_s)$ be the primal solution output by A_{online}, which is a function of $\mathbf{p}_s$. We have

$$x_{i\eta}(\mathbf{p}_s) = \begin{cases} 1, \text{ if } \eta = \arg\max\limits_{\eta' \in [N_i]} \{b_{i\eta'} - \sum\limits_{k\in[K]} \frac{\omega^k_{i\eta'}}{T} p_{k,s}\} \text{ and } b_{i\eta} > \sum\limits_{k\in[K]} \frac{\omega^k_{i\eta}}{T} p_{k,s}, \\ 0, \text{ otherwise.} \end{cases} \tag{4.12}$$

We next define two random variables $\mathbf{X}^k_i$ and $\mathbf{Y}^k_i(t)$, which will be used in the following analysis.

$$\mathbf{X}^k_i = \sum_{\eta\in[N_i]} \frac{\omega^k_{i\eta}}{T} x_{i\eta}(\mathbf{p}_s). \tag{4.13}$$

$$\mathbf{Y}^k_i(t) = \begin{cases} \sum\limits_{l\in\zeta_i} f_{il}(t) x_{il}, \text{ if } t \in T(l), \\ 0, \text{ otherwise.} \end{cases} \tag{4.14}$$

Note that the value of x_{il} in $\mathbf{Y}^k_i(t)$ is output by A_{online} and computed according to the value of $x_{i\eta}(\mathbf{p}_s)$.

Lemma 14. The expectation of $\mathbf{Y}^k_i(t)$ on t is upper bounded by $\mathbf{X}^k_i$ when job i's arrival time t_i is uniformly disturbed in $[T]$.

Proof: If job i is rejected by the cloud provider, *i.e.,* $x_{i\eta}(\mathbf{p}_s) = 0, \forall \eta \in [N_i]$, then $\mathbf{Y}^k_i(t) = \mathbf{X}^k_i$ and the lemma follows. Otherwise, let $\eta_i = \arg\max_{\eta\in[N_i]} x_{i\eta}(\mathbf{p}_s)$. According to A_{online}, job i is scheduled within $[a_i, a_i + \sum_{n=1}^{\eta_i} L_{in})$, and let l_i be the corresponding schedule. For a fixed $t \in [T]$, we have

$$E(\mathbf{Y}^k_i(t)) = Pr[a_i \le t < a_i + \sum_{n=1}^{\eta_i} L_{in})] f^k_{il_i}(t) = Pr[t - \sum_{n=1}^{\eta_i} L_{in} < a_i \le t] f^k_{il_i}(t).$$

Because a_i is uniformly disturbed in $[T]$, $Pr[a_i = t] = \frac{1}{T}$. Moreover, since $1 \leq t \leq T$, $Pr[t - \sum_{n=1}^{\eta_i} L_{in} < a_i \leq t]$ has two different values when $1 \leq t < \sum_{n=1}^{\eta_i} L_{in}$ and $\sum_{n=1}^{\eta_i} L_{in} \leq t \leq T$. For both cases, we have

$$E(\mathbf{Y}_i^k(t)) \leq \frac{1}{T}\sum_{n=1}^{\eta_i} r_{in}^k L_{in} = \frac{\omega_{i\eta_i}^k}{T} = \mathbf{X}_i^k. \qquad \square$$

Lemma 15. Let E_1 denote the event that the total number of jobs that arrived in $[T]$, I, is within the range of $[(1-\frac{\mathscr{F}_s}{2})\lambda T, (1+\frac{\mathscr{F}_s}{2})\lambda T]$, $\forall \mathscr{F}_s$. The probability that E_1 happens is at least $1-\varepsilon_2$, given $\lambda T \geq \frac{4}{(\varepsilon_2)^3}$.

Proof:

$$Pr[E_1] \geq 1 - Pr[|I - \lambda T| \geq \frac{\mathscr{F}_s}{2}\lambda T] \geq 1 - Pr[|I - \lambda T| \geq \frac{\varepsilon_2}{2}\lambda T].$$

The last inequality holds because $\mathscr{F}_s \geq \varepsilon_2$. According to Chebyshev's inequality [59], we can obtain

$$Pr[|I - \lambda T| \geq \frac{\varepsilon_2}{2}\lambda T] = Pr[|I - E[I]| \geq \frac{\varepsilon_2}{2}\lambda T] \leq \frac{Var[I]}{(\frac{\varepsilon_2}{2}\lambda T)^2} = \frac{4\lambda T}{\varepsilon_2^2 \lambda^2 T^2} = \frac{4}{\varepsilon_2^2 \lambda T}.$$

Given $\lambda T \geq \frac{4}{(\varepsilon_2)^3}$, we have $\frac{4}{\varepsilon_2^2 \lambda T} \leq \varepsilon_2$ and therefore $Pr[E_1] \geq 1 - \varepsilon_2$. $\square$

We define a new variable $\mathscr{B}$, and let

$$\mathscr{B} = \max\left\{\frac{12\ln\left(2(IN)^K KT\log_2(\frac{1}{\varepsilon_2})/\varepsilon_2\right)}{\varepsilon_2^2}, \frac{4\lambda T}{\varepsilon_2^2}\right\}.$$

Lemma 16. Let E_2 denote the event that

$$\sum_{i \in \mathscr{I}_{s+1} \setminus \mathscr{I}_s} \mathbf{X}_i^k \geq 2^s \varepsilon_2 c_k, \forall k \in [K], s \in [0, 1, \ldots, S].$$

On the condition of E_1, *i.e.,* $(1-\frac{\mathscr{F}_s}{2})\lambda T \leq I \leq (1+\frac{\mathscr{F}_s}{2})\lambda T$, the probability that E_2 happens, $Pr[E_2|E_1]$, is at most $\frac{\varepsilon_2}{T}$, given $\frac{c_k}{r_{max}^k} \geq \mathscr{B}$.

Proof: Consider a fixed price $\mathbf{p}$. We say a random sample $\mathscr{I}_{s+1} \setminus \mathscr{I}_s$ is bad for this $\mathbf{p}$ if $\mathbf{p} = \mathbf{p}_s$ but $\sum_{i \in \mathscr{I}_{s+1} \setminus \mathscr{I}_s} \sum_{\eta \in [N_i]} \frac{\omega_{i\eta}^k}{T} x_{i\eta}(\mathbf{p}) \geq 2^s \varepsilon_2 c_k$, for some k and s. We first show that the probability of

bad samples is small for every fixed $\mathbf{p}$, s and k. Then we take union bound over all "distinct" prices, all s, and all k to prove with small probability, $\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s}\sum_{\eta\in[N_i]}\frac{\omega_{i\eta}^k}{T}x_{i\eta}(\mathbf{p}_s)\geq 2^s\varepsilon_2 c_k, \forall k, \forall s$ with price $\mathbf{p}_s$.

We first fix $\mathbf{p}$, s and k. Recall the definition of $\mathbf{X}_i^k$ in (4.12). Since $\mathbf{p_s}$ is the optimal solution for LP (4.11), then by complementary conditions, we have $\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k\leq(1-\mathscr{F}_s)2^s\varepsilon_2 c_k$. We define events

$$A=\{\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k\leq(1-\mathscr{F}_s)2^s\varepsilon_2 c_k\}, B=\{\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s}\mathbf{X}_i^k\geq 2^s\varepsilon_2 c_k\}.$$

Therefore, the probability of bad samples is bounded by:

$$\begin{aligned}Pr[B]&=Pr[\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k-\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k\geq 2^s\varepsilon_2 c_k]=Pr[\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k\geq(2-\mathscr{F}_s)2^s\varepsilon_2 c_k|A]\\&\leq Pr[|\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k-\frac{I_s}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k|\geq\beta]\end{aligned}\tag{4.15}$$

Because $\frac{I_s}{I_{s+1}}=\frac{1}{2}\geq\frac{1}{2(1+\mathscr{F}_s/2)}$ or $\frac{I_s}{I_{s+1}}=\frac{\lambda T/2}{I}\geq\frac{1}{2(1+\mathscr{F}_s/2)}$ as $I\leq(1+\frac{\mathscr{F}_s}{2})\lambda T$, thus,

$$\begin{aligned}&|\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k-\frac{I_s}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k|\geq(\frac{1}{1+\mathscr{F}_s/2}(1-\mathscr{F}_s/2)-(1-\mathscr{F}_s))2^s\varepsilon_2 c_k\\&=\frac{\frac{\mathscr{F}_s^2}{2}}{1+\mathscr{F}_s/2}2^s\varepsilon_2 c_k\geq\frac{\mathscr{F}_s^2}{4}2^s\varepsilon_2 c_k.\end{aligned}$$

Then $\beta=\frac{\mathscr{F}_s^2}{4}2^s\varepsilon_2 c_k$.

We normalize r_{max}^k such that $\mathbf{X}_i^k\in[0,1]$, and replace c_k with $\frac{c_k}{r_{max}^k}$. We define random variables:

$$\sigma^2(\mathbf{X})=\frac{1}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}(\mathbf{X}_i^k-\frac{1}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k)^2\leq 1.$$

$$\Delta(\mathbf{X})=\max_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k-\min_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k\leq 1.$$

According to Hoeffding-Berstein Inequality (Appendix A.1 in [9]), we have

$$(4.15)\leq 2\exp\left(-\frac{\beta^2}{2I_s\sigma^2(\mathbf{X})+\beta\Delta(\mathbf{X})}\right)\leq 2\exp\left(-\frac{\frac{\mathscr{F}_s^4}{16}2^{2s}\varepsilon_2^2c_k^2}{2I_s+\frac{\mathscr{F}_s^2}{4}2^s\varepsilon_2 c_k}\right).\tag{4.16}$$

$I_s\leq 2^s\varepsilon_2\lambda T\leq\lambda T$. Because $c_k/r_{max}^k=c_k\geq 4\lambda T/\varepsilon_2^2$, we have $I_s\leq\lambda T\leq\varepsilon_2^2c_k/4=\frac{\mathscr{F}_s^2}{4}2^s\varepsilon_2 c_k$. Thus,

$$(4.16)\leq 2\exp\left(-\frac{\frac{\mathscr{F}_s^2}{4}2^s\varepsilon_2 c_k}{2+1}\right)\leq 2\exp\left(-\frac{\varepsilon_2^2c_k}{12}\right)\leq\frac{\varepsilon_2}{K(IN)^KT\log_2(\frac{1}{\varepsilon_2})}.$$

The last inequality holds because $c_k/r^k_{max} = c_k \geq \mathscr{B}$. Next, we take a union bound over all "distinct" $\mathbf{p}$. Two price vectors $\mathbf{p_1}$ and $\mathbf{p_2}$ are distinct if and only if they result in distinct solution, *i.e.,* $x_{i\eta}(\mathbf{p_1}) \neq x_{i\eta}(\mathbf{p_2})$. By results from computational geometry [58], the total number of such distinct prices is at most $(IN)^K$. Taking the union bound over all distinct prices, K types of resources and $\log_2(\frac{1}{\varepsilon_2})$ stages, we get the desired result. □

Lemma 17. With probability at least $1-2\varepsilon_2$, we have

$$\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} E[\mathbf{Y}^k_i(t)] \leq 2^s\varepsilon_2 c_k, \forall k \in [K], t \in [T], s \in [0,1,\ldots,S],$$

given $\lambda T \geq \frac{4}{(\varepsilon_2)^3}$ and $\frac{c_k}{r^k_{max}} \geq \mathscr{B}$.

Proof: We first prove that, for a fixed t, on the condition of E_1, the probability of $\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} E[\mathbf{Y}^k_i(t)] \geq 2^s\varepsilon_2 c_k, \forall k, \forall s$ is small.

According to Lemma 14, the expectation of $\mathbf{Y}^k_i(t)$ on t is upper bounded by $\mathbf{X}^k_i$. Therefore,

$$\begin{aligned}
&Pr[\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} E[\mathbf{Y}^k_i(t)] \geq 2^s\varepsilon_2 c_k, \forall k, \forall s|E_1] \leq Pr[\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} \mathbf{X}^k_i \geq 2^s\varepsilon_2 c_k \forall k, \forall s|E_1] \\
&= Pr[E_2|E_1] \leq \frac{\varepsilon_2}{T}. \text{ (Lemma 16)}
\end{aligned}$$

We take the union bound over T slots and have

$$\begin{aligned}
&P[\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} E[\mathbf{Y}^k_i(t)] \leq 2^s\varepsilon_2 c_k, \forall k, \forall s, \forall t] \geq P[\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} E[\mathbf{Y}^k_i(t)] \leq 2^s\varepsilon_2 c_k, \forall k, \forall s, \forall t|E_1]Pr[E_1] \\
&\geq (1 - T\cdot Pr[\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} E[\mathbf{Y}^k_i(t)] \geq 2^s\varepsilon_2 c_k, \forall k, \forall s|E_1])Pr[E_1] \\
&\geq (1-\varepsilon_2)^2 \geq 1-2\varepsilon_2. \text{ (Lemma 15)}
\end{aligned}$$

□

iii) Competitive Ratio.

Finally, we show that our algorithm A_{online} is $\frac{1}{1-O(\varepsilon_2)}$-competitive in expectation in Theorem 9.

Lemma 18. Let OPT denote the optimal objective value of the offline problem in (4.2). $E[OPT]$ is the expectation of OPT over all possible realizations of the job arrival process. The optimal objective value of LP (4.8) is at least $E[OPT]$.

Proof: We observe that the average of the optimal solutions of the offline problem in (4.2), computed over all possible realizations of the job arrival process, achieves the expected offline social welfare $E[OPT]$. Furthermore, it also provides a feasible solution to the expected offline problem in (4.8). Therefore, the optimal objective value of LP (4.8) must be at least $E[OPT]$. $\square$

As $|\mathscr{I}_s| \leq 2^S \lfloor \varepsilon_2 \lambda T \rfloor \leq \frac{\lambda T}{2} < \lambda T$, we have the following observation.

Observation 1. *The inputs of the problem in (4.10) have the following property: for the optimal dual solution* $\mathbf{p}_s$ *derived by solving the dual problem (4.11), there can be at most* λT *equations such that* $b_{i\eta_i} = \sum_{k\in[K]} \frac{\omega_{i\eta_i}}{k} p_{k,s}, \forall i \in \mathscr{I}_s$*, where* η_i *denotes the best option for job i.*

Lemma 19. Let $\{x_{i\eta,s}\}_{i\in[I],\eta\in[N_i]}$ be the optimal solution of (4.10), and $\mathbf{x}_s$ be the solution vector.

$$\sum_{i\in\mathscr{I}_s}\sum_{\eta\in[N_i]} x_{i\eta,s} - \lambda T \leq \sum_{i\in\mathscr{I}_s}\sum_{\eta\in[N_i]} x_{i\eta}(\mathbf{p}_s) \leq \sum_{i\in\mathscr{I}_s}\sum_{\eta\in[N_i]} x_{i\eta,s}, \forall s \in [0,1,\ldots,S].$$

Proof: Let η_i denote the best option for job i, *i.e.,* $\eta_i = \arg\max_{\eta'\in[N_i]}\{b_{i\eta'} - \sum_{k\in[K]} \frac{\omega^k_{i\eta'}}{T} p_{k,s}\}$. By complementary slackness, the optimal solution of (4.10) satisfies $x_{i\eta_i,s} = 0$ if $b_{i\eta_i} < \sum_{k\in[K]} \frac{\omega^k_{i\eta_i}}{T} p_{k,s}$, and $x_{i\eta_i,s} > 0$ if $b_{i\eta_i} = \sum_{k\in[K]} \frac{\omega^k_{i\eta_i}}{T} p_{k,s}$. Compared with (4.12), the only difference is $x_{i\eta_i}(\mathbf{p}_s) = 0$ when $b_{i\eta_i} = \sum_{k\in[K]} \frac{\omega^k_{i\eta_i}}{T} p_{k,s}$. These imply that jobs accepted by A_{online} are also accepted by the optimal solution, while some jobs rejected by A_{online} are accepted by the optimal solution. Since Observation 1 indicates that there are at most λT equations that satisfy $b_{i\eta_i} = \sum_{k\in[K]} \frac{\omega_{i\eta_i}}{k} p_{k,s}, \forall i \in \mathscr{I}_s$, there are at most λT jobs that are rejected by A_{online} but accepted by the optimal solution. $\square$

Lemma 20. On the condition of $(1 - \frac{\mathscr{F}_s}{2})\lambda T \leq I \leq (1 + \frac{\mathscr{F}_s}{2})\lambda T$, with probability at least $1 - \varepsilon_2$, $\forall s \in [0,\ldots,S]$,

$$\sum_{i\in\mathscr{I}_{s+1}}\sum_{\eta\in[N_i]} b_{i\eta} x_{i\eta}(\mathbf{p}_s) \geq (1 - 3\mathscr{F}_s) P^*_{s+1}(\mathbf{x}_{s+1}),$$

where $\sum_{i\in\mathscr{I}_{s+1}}\sum_{\eta\in[N_i]} b_{i\eta} x_{i\eta}(\mathbf{p}_s)$ is the objective value of P_{s+1} in (4.10) achieved by our solution $x_{i\eta}(\mathbf{p}_s)$, and $P^*_{s+1}(\mathbf{x}_{s+1})$ is the optimal objective value of P_{s+1} in (4.10) under optimal solution $\mathbf{x}_{s+1}$, given $\frac{c_k}{r^k_{max}} \geq \mathscr{B}$.

Proof: We first define an auxiliary primal problem as follows:

$$P_A: \quad \text{maximize} \sum_{i\in\mathscr{I}_{s+1}} \sum_{\eta\in[N_i]} b_{i\eta} x_{i\eta} \tag{4.17}$$

subject to:

$$\sum_{i\in\mathscr{I}_{s+1}} \sum_{\eta\in[N_i]} \frac{\omega_{i\eta}^k}{T} x_{i\eta} \leq A_k, \forall k \in [K], \tag{4.17a}$$

$$\sum_{\eta\in[N_i]} x_{i\eta} \leq 1, \forall i \in \mathscr{I}_{s+1}, \tag{4.17b}$$

$$x_{i\eta} \geq 0, \forall i \in \mathscr{I}_{s+1}, \forall \eta \in [N_i]. \tag{4.17c}$$

where $A_k = \sum_{i\in\mathscr{I}_{s+1}} \sum_{\eta\in[N_i]} \frac{\omega_{i\eta}^k}{T} x_{i\eta}(\mathbf{p}_s)$ if $p_{k,s} > 0$ and $A_k = \max\{\sum_{i\in\mathscr{I}_{s+1}} \sum_{\eta\in[N_i]} \frac{\omega_{i\eta}^k}{T} x_{i\eta}(\mathbf{p}_s), 2^{s+1}\varepsilon_2 c_k\}$ if $p_{k,s} = 0$. Its dual problem is:

$$D_A: \quad \text{minimize} \sum_{k\in[K]} A_k c_k p_k + \sum_{i\in\mathscr{I}_{s+1}} u_i \tag{4.18}$$

subject to:

$$u_i \geq b_{i\eta} - \sum_{k\in[K]} \frac{\omega_{i\eta}^k}{T} p_k, \forall i \in \mathscr{I}_{s+1}, \forall \eta \in [N_i], \tag{4.18a}$$

$$p_k, u_i \geq 0, \forall k \in [K], \forall i \in \mathscr{I}_{s+1}. \tag{4.18b}$$

Note that $\{x_{i\eta}(\mathbf{p}_s)\}_{i\in[i],\eta\in[N_i]}$ and $\mathbf{p}_s$ satisfy all complementarity conditions, and therefore they are the optimal primal and dual solutions to LP (4.17) and LP (4.18). The optimal objective value of (4.17) is $\sum_{i\in\mathscr{I}_{s+1}} \sum_{\eta\in[N_i]} b_{i\eta} x_{i\eta}(\mathbf{p}_s)$. In order to prove the lemma, we need to show that with probability at least $1-\varepsilon_2$, $(1-3\mathscr{F}_s)\mathbf{x}_{s+1}$ is a feasible solution to auxiliary program (4.17).

First, we show that with probability at least $1-\varepsilon_2$,

$$A_k \geq (1-3\mathscr{F}_s)2^{s+1}\varepsilon_2 c_k, \forall k \in [K], s \in [0,\ldots,S]. \tag{4.19}$$

If $p_{k,s} = 0$, then by definition we have $A_k \geq 2^{s+1}\varepsilon_2 c_k$. It remains to prove the case where $p_{k,s} > 0$ that, with probability at least $1-\varepsilon_2$, $A_k \geq (1-3\mathscr{F}_s)2^{s+1}\varepsilon_2 c_k, \forall k \in [K], s \in [0,\ldots,S]$. This

is proven by showing that with probability at most ε_2, $A_k = \sum_{i\in\mathscr{I}_{s+1}}\sum_{\eta\in[N_i]}\frac{\omega_{i\eta}^k}{T}x_{i\eta}(\mathbf{p}_s) \le (1-3\mathscr{F}_s)2^{s+1}\varepsilon_2 c_k, \forall k\in[K], s\in[0,\dots,S]$. The detailed proof is as follows: Recall that $\{x_{i\eta,s}\}_{i\in[I],\eta\in[N_i]}$ and $\{p_{k,s}\}_{k\in[K]}$ are the optimal solutions to programs (4.10) and (4.11). Then, by complementary slackness, if $p_{k,s}>0$, we have $\sum_{i\in\mathscr{I}_{s+1}}\sum_{\eta\in[N_i]}\frac{\omega_{i\eta}^{k,s}}{T}x_{i\eta,s} = (1-\mathscr{F}_s)2^s\varepsilon_2 c_k$. We normalize r_{max}^k such that $r_{max}^k = 1$. Given $c_k/r_{max}^k \ge \frac{4\lambda T}{\varepsilon_2^2} \ge \frac{\lambda T}{2^s\varepsilon_2^2}$, and the observation in Lemma 19, we have for any k and s,

$$\sum_{i\in\mathscr{I}_s}\sum_{\eta\in[N_i]}\frac{\omega_{i\eta}^k}{T}x_{i\eta}(\mathbf{p}_s) \ge \sum_{i\in\mathscr{I}_s}\sum_{\eta\in[N_i]}\frac{\omega_{i\eta}^k}{T}x_{i\eta,s} - \lambda T$$
$$\ge (1-\mathscr{F}_s-\varepsilon_2)2^s\varepsilon_2 c_k \ge (1-2\mathscr{F}_s)2^s\varepsilon_2 c_k.$$

For a fixed k, s and a distinct price vector $\mathbf{p}$, when $\mathbf{p}=\mathbf{p}_s$, we define events $G_1 = \{\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k \le (1-3\mathscr{F}_s)2^{s+1}\varepsilon_2 c_k\}$ and $G_2 = \{\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k \ge (1-2\mathscr{F}_s)2^s\varepsilon_2 c_k\}$.

$$Pr[G_1] = Pr[G_1|G_2] \le Pr[|\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k - \frac{I_s}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k| \ge \beta'].\tag{4.20}$$

Because $\frac{I_s}{I_{s+1}} = \frac{1}{2} \le \frac{1}{2(1-\mathscr{F}_s/2}$) or $\frac{I_s}{I_{s+1}} = \frac{\lambda T/2}{I} \le \frac{1}{2(1-\mathscr{F}_s/2)}$ as $I \ge (1-\frac{\mathscr{F}_s}{2})\lambda T$, thus,

$$\sum_{i\in\mathscr{I}_s}\mathbf{X}_i^k - \frac{I_s}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k \ge (1-2\mathscr{F}_s - \frac{1}{1-\mathscr{F}_s/2}(1-3\mathscr{F}_s))2^s\varepsilon_2 c_k$$
$$= \frac{\mathscr{F}_s+2\mathscr{F}_s^2}{2-\mathscr{F}_s}2^s\varepsilon_2 c_k \ge \frac{\mathscr{F}_s}{2}2^s\varepsilon_2 c_k.$$

Then $\beta' = \frac{\mathscr{F}_s}{2}2^s\varepsilon_2 c_k$. Note that $\mathbf{X}_i^k \in [0,1]$ as $r_{max}^k = 1$. Next, similar to the proof of Lemma 16, we define two random variables:

$$\sigma^2(\mathbf{X}) = \frac{1}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}(\mathbf{X}_i^k - \frac{1}{I_{s+1}}\sum_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k)^2 \le 1.$$

$$\Delta(\mathbf{X}) = \max_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k - \min_{i\in\mathscr{I}_{s+1}}\mathbf{X}_i^k \le 1.$$

According to Hoeffding-Berstein Inequality [9], we have

$$(4.20) \le 2\exp\left(-\frac{\beta'^2}{2I_s\sigma^2(\mathbf{X})+\beta\Delta(\mathbf{X})}\right) \le 2\exp\left(-\frac{\frac{\mathscr{F}_s^2}{4}2^{2s}\varepsilon_2^2 c_k^2}{2I_s+\frac{\mathscr{F}_s}{2}2^s\varepsilon_2 c_k}\right).\tag{4.21}$$

Because $c_k/r^k_{max} = c_k \geq \lambda T$, we have $2I_s \leq 2 \cdot 2^s \varepsilon_2 \lambda T \leq 2 \cdot 2^s \varepsilon_2 c_k$. Hence,

$$(4.21) \leq 2\exp\left(-\frac{\frac{\mathscr{F}_s^2}{4} 2^s \varepsilon_2 c_k}{2+\frac{\mathscr{F}_s}{2}}\right) \leq 2\exp\left(-\frac{\varepsilon_2^2 c_k}{12}\right) \leq \frac{\varepsilon_2}{K(IN)^K \log_2(\frac{1}{\varepsilon_2})}.$$

The last inequality holds because $c_k/r^k_{max} = c_k \geq \mathscr{B}$. Taking the union bound over $(IN)^K$ distinct prices, K types of resources and $\log_2(\frac{1}{\varepsilon_2})$ stages, we prove that with probability at least $1-\varepsilon_2$, $A_k \geq (1-3\mathscr{F}_s)2^{s+1}\varepsilon_2 c_k, \forall k \in [K], s \in [0,\ldots,S]$.

We observe that i) constraints (4.10b) and (4.10c) are the same as (4.17b) and (4.17c); ii) constraints (4.10a) and (4.17a) only differ in the RHS. Following the result of (4.19), we have with probability at least $1-\varepsilon_2$, $(1-3\mathscr{F}_s)\mathbf{x}_{s+1}$ is a feasible solution to LP (4.17). Therefore, with probability at least $1-\varepsilon_2$, the optimal objective value of (4.17), *i.e.*, $\sum_{i\in\mathscr{I}_{s+1}}\sum_{\eta\in[N_i]} b_{i\eta}x_{i\eta}(\mathbf{p}_s)$, is at least the objective value of (4.17) under the solution $(1-3\mathscr{F}_s)\mathbf{x}_{s+1}$, *i.e.,* $(1-3\mathscr{F}_s)P^*_{s+1}(\mathbf{x}_{s+1})$. □

Lemma 21.

$$E[P^*_s(\mathbf{x}_s)] \leq 2^s \varepsilon_2 P^*_\Sigma, \forall s \in [0,\ldots,S],$$

where $E(P^*_s(\mathbf{x}_s))$ is the expectation of the optimal objective value of P_s in (4.10) achieved by the optimal solution $\mathbf{x}_s$ over all possible realizations of the job arrival process, and P^*_Σ is the optimal objective value of (4.8).

Proof: Let $(\mathbf{x}_s, \mathbf{p}_s, \mathbf{u}_s)$ denote the optimal primal-dual solution to (4.10) and (4.11), and $(\mathbf{x}_\Sigma, \mathbf{p}_\Sigma, \mathbf{u}_\Sigma)$ denote the optimal primal-dual solution to (4.8) and (4.9). Comparing the two dual programs (4.9) and (4.11), we can observe that $(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma)$ is a feasible solution to program D_s in (4.11) since any realization of job $i \in \mathscr{I}_s$ can be found in the distribution $\mathscr{D}$. Then the objective value of (4.11) with solution $(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma)$, $D_s(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma)$, is at least the optimal objective value $D^*_s(\mathbf{p}_s, \mathbf{u}_s)$. Furthermore, according to weak duality, $P^*_s(\mathbf{x}_s) \leq D^*_s(\mathbf{p}_s, \mathbf{u}_s) \leq D_s(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma)$. Then we have

$$E[P^*_s(\mathbf{x}_s)] \leq E[D^*_s(\mathbf{p}_s, \mathbf{u}_s)] \leq E[D_s(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma)] = E[\sum_{k\in[K]}(1-\mathscr{F}_s)2^s\varepsilon_2 c_k p_{k,\Sigma} + \sum_{i\in\mathscr{I}_s} u_{i,\Sigma}]$$

$$\leq E[2^s\varepsilon_2 \sum_{k\in[K]} c_k p_{k,\Sigma} + \sum_{i\in\mathscr{I}_s} u_{i,\Sigma}] \leq 2^s\varepsilon_2 \sum_{k\in[K]} c_k p_{k,\Sigma} + \sum_{j\in\mathscr{D}} I_s \rho_j u_{j,\Sigma}$$

$$\leq 2^s\varepsilon_2(\sum_{k\in[K]} c_k p_{k,\Sigma} + \sum_{j\in\mathscr{D}} \lambda T\rho_j u_{j,\Sigma}) = 2^s\varepsilon_2 D^*_\Sigma(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma) = 2^s\varepsilon_2 P^*_\Sigma. \qquad \square$$

Lemma 22. On the condition of $(1-\frac{\mathscr{F}_s}{2})\lambda T \leq I \leq (1+\frac{\mathscr{F}_s}{2})\lambda T$,

$$(1-\varepsilon_2)P^*_\Sigma \leq E[P^*_{S+1}(\mathbf{x}_{S+1})] \leq (1+\mathscr{F}_s/2)P^*_\Sigma,$$

where $S+1 = \log_2(\frac{1}{\varepsilon_2})$, $E(P^*_{S+1}(\mathbf{x}_{S+1}))$ is the expectation of the optimal objective value of P_{S+1} and P^*_Σ is the optimal objective value of (4.8).

Proof: We first prove that $E[P^*_{S+1}(\mathbf{x}_{S+1})] \leq (1+\mathscr{F}_s/2)P^*_\Sigma$. Similar to the proof in Lemma 21, we have

$$\begin{aligned}
&E[P^*_{S+1}(\mathbf{x}_{S+1})] \leq E[D^*_{S+1}(\mathbf{p}_{S+1}, \mathbf{u}_{S+1})] \leq E[D_{S+1}(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma)] \\
&\leq E[\sum_{k\in[K]} c_k p_{k,\Sigma} + \sum_{i\in[I]} u_{i,\Sigma}] \leq \sum_{k\in[K]} c_k p_{k,\Sigma} + \sum_{j\in\mathscr{D}} (1+\frac{\mathscr{F}_s}{2})\lambda T\rho_j u_{j,\Sigma} \\
&\leq (1+\frac{\mathscr{F}_s}{2})(\sum_{k\in[K]} c_k p_{k,\Sigma} + \sum_{j\in\mathscr{D}} \lambda T\rho_j u_{j,\Sigma}) = (1+\frac{\mathscr{F}_s}{2})D^*_\Sigma(\mathbf{p}_\Sigma, \mathbf{u}_\Sigma) = (1+\frac{\mathscr{F}_s}{2})P^*_\Sigma.
\end{aligned}$$

Next, we show $(1-\varepsilon_2)P^*_\Sigma \leq E[P^*_{S+1}(\mathbf{x}_{S+1})]$. When $S+1 = \log_2(\frac{1}{\varepsilon_2})$, constraints (4.10a) in program P_{S+1} becomes $\sum_{i\in\mathscr{I}_s}\sum_{\eta\in[N_i]} \frac{\omega^k_{i\eta}}{T} x_{i\eta} \leq (1-\varepsilon_2)c_k$. Consider a new version of LP (4.8) by replacing constraints (4.8a) with $\sum_{j\in\mathscr{D}}\sum_{\eta\in[N_j]} \lambda T\rho_j \frac{\omega^k_{j\eta}}{T} x_{j\eta} \leq (1-\varepsilon_2)c_k$, and denote this new program by $P_{\Sigma'}$. Let $P^*_{\Sigma'}$ be the optimal objective value of $P_{\Sigma'}$, and $\mathbf{x}_\Sigma$ be the optimal solution of (4.8). Then $(1-\varepsilon_2)\mathbf{x}_\Sigma$ must be a feasible solution to $P_{\Sigma'}$, and the objective value under this solution is at most $P^*_{\Sigma'}$, *i.e.,* $P_{\Sigma'}(\mathbf{x}_\Sigma) = (1-\varepsilon_2)P^*_\Sigma \leq P^*_{\Sigma'}$. In addition, comparing P_{S+1} and $P_{\Sigma'}$, we found the expectation of optimal objective value of P_{S+1} is equal to $P^*_{\Sigma'}$. Therefore, $E[P^*_{S+1}(\mathbf{x}_{S+1})] = P^*_{\Sigma'} \geq (1-\varepsilon_2)P^*_\Sigma$. $\square$

Theorem 9. For any $0 < \varepsilon_2 < \frac{1}{23}$, our online scheduling algorithm A_{online} is $\frac{1}{(1-23\varepsilon_2)}$-competitive in expectation with *i.i.d.* job types and uniform job arrival time distribution, as compared to the expected optimal objective value of offline problem in (4.2), given $\lambda T \geq \frac{4}{(\varepsilon_2)^3}$ and $\frac{c_k}{r^k_{max}} \geq \mathscr{B}$.

Proof: Combining Lemma 15, Lemma 17, and Lemma 20, we have with probability at least $(1-\varepsilon_2)\times(1-\varepsilon_2)\times(1-2\varepsilon_2)\geq 1-4\varepsilon_2$, events

$$(1-\frac{\mathscr{F}_s}{2})\lambda T \leq I \leq (1+\frac{\mathscr{F}_s}{2})\lambda T,$$

$$\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s} E[\mathbf{Y}_i^k(t)] \leq 2^s\varepsilon_2 c_k,$$

$$\sum_{i\in\mathscr{I}_{s+1}}\sum_{\eta\in[N_i]} b_{i\eta}x_{i\eta}(\mathbf{p}_s) \geq (1-3\mathscr{F}_s)P^*_{s+1}(\mathbf{x}_{s+1}),$$

happen simultaneously for all $k\in[K], t\in[T]$ and $s\in[0,\ldots,S]$. Let Ψ denote the event that three events happen simultaneously. Then we can have:

$$\begin{aligned}
&E[\sum_{s=0}^{S}\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s}\sum_{\eta\in[N_i]} b_{i\eta}x_{i\eta}(\mathbf{p}_s)|\Psi] \\
&\geq E[\sum_s\sum_{i\in\mathscr{I}_{s+1}}\sum_{\eta\in[N_i]} b_{i\eta}x_{i\eta}(\mathbf{p}_s)|\Psi] - E[\sum_s\sum_{i\in\mathscr{I}_s}\sum_{\eta\in[N_i]} b_{i\eta}x_{i\eta}(\mathbf{p}_s)|\Psi] \\
&\geq \sum_s(1-3\mathscr{F}_s)E[P^*_{s+1}(\mathbf{x}_{s+1})|\Psi] - \sum_s E[P^*_s(\mathbf{x}_s)|\Psi] \qquad (4.22)
\end{aligned}$$

Combining Lemma 21 and Lemma 22, we have

$$\begin{aligned}
(4.22) &\geq (1-\varepsilon_2)P^*_\Sigma - \frac{1}{Pr[\Psi]}\left(E[P^*_0(\mathbf{x}_0)] + \sum_s 3\mathscr{F}_s E[P^*_{s+1}(\mathbf{x}_{s+1})]\right) \\
&\geq (1-\varepsilon_2)P^*_\Sigma - \frac{1}{1-4\varepsilon_2}\left(\varepsilon_2 + \sum_{s=0}^{S-1} 3\varepsilon\mathscr{F}_s 2_2^{s+1} + 3\mathscr{F}_S(1+\frac{\mathscr{F}_s}{2})\right)P^*_\Sigma \\
&\geq (1-\varepsilon_2)P^*_\Sigma - \frac{1}{1-4\varepsilon_2}(1+6\times 1.8+3\times\sqrt{2}\times(1+\frac{\sqrt{0.5}}{2})\varepsilon_2 P^*_\Sigma \\
&\geq (1-\varepsilon_2)P^*_\Sigma - \frac{1}{1-4\varepsilon_2}18\varepsilon_2 P^*_\Sigma.
\end{aligned}$$

The last two inequalities hold because $\sum_{s=0}^{S-1}\mathscr{F}_s 2^s\varepsilon_2 \leq 1.8\varepsilon_2, \mathscr{F}_s \leq \sqrt{\varepsilon_2} \leq \sqrt{0.5}$.

$$\begin{aligned}
&E[\sum_{s=0}^{S}\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s}\sum_{\eta\in[N_i]} b_{i\eta}x_{i\eta}(\mathbf{p}_s)|\Psi] \geq Pr[\Psi]\times E[\sum_{s=0}^{S}\sum_{i\in\mathscr{I}_{s+1}\setminus\mathscr{I}_s}\sum_{\eta\in[N_i]} b_{i\eta}x_{i\eta}(\mathbf{p}_s)] \\
&\geq (1-4\varepsilon_2)\left((1-\varepsilon_2)P^*_\Sigma - \frac{1}{1-4\varepsilon_2}18\varepsilon_2 P^*_\Sigma\right) \geq (1-23\varepsilon_2)P^*_\Sigma \geq (1-23\varepsilon_2)E[OPT]. \qquad \square
\end{aligned}$$

4.5.4 Discussion

A_{online} can be generalized to handle general jobs with arbitrary dependence graph topology. Upon the arrival of the ith job, we first compute a topological ordering of its dependence graph. Such ordering ensures that if job i's subtask j must be executed before subtask k, j precedes k in the ordering. It can be accomplished in linear time, *e.g.,* by Kahn's algorithm or depth-first search [71]. We then re-index its subtasks according to the output ordering. The rest of the algorithm design is the same as the counterpart in Ch. 4.5.2, and we omit the details. Because the expected offline optimization problem for general jobs can also be formulated to LP (4.8) and our online algorithm design is based on this LP, the online algorithm for general jobs can achieve the same performance as A_{online} does, with regard to optimality and feasibility. The theoretical analysis is similar to the counterpart in Ch. 4.5.3, and is omitted here.

4.6 Performance Evaluation

In this section, we evaluate our offline and online scheduling algorithms through trace-driven simulation studies. We further compare our scheduling algorithms with two related algorithms from the recent literature [44] [84]. They study the similar cloud scheduling problem under simplified offline and online scenarios by assuming that each job contains only one subtask. We first introduce the simulation setup. We configure each job according to Google Cluster Data (version 1 [3]) which contains each job's information including number of subtasks, execution duration, and resource demands (CPU and RAM). We assume each subtask occupies $[1,12]$ slots, and each slot is 5 minutes. By default, the maximum number of subtasks (N) is 5, $\lambda = 0.5$ and $T = 500$. The total number of jobs I is decided according to a Poisson distribution with expectation of λT. The arrival time of each job is independently and uniformly chosen within $[1,T]$ to simulate a Poisson process. Each job's deadline is also generated uniformly at random between its arrival time and T. The value of each subtask (b_{in}) is computed as its overall resource demand times unit prices randomly picked in the range $[1,50]$. The capacity of each type of resource is normalized to 1.

The default value of $C = \min_{k \in [K]} \{ \frac{c_k}{r^k_{max}} \}$ in our experiments is 1, which is much smaller than the value in our assumption. Although a lower bound of C is required for our theoretical analysis, it can be observed that even when the assumption is violated, our offline and online algorithms can still achieve a close-to-optimal performance in practice.

4.6.1 Performance of $A_{offline}$

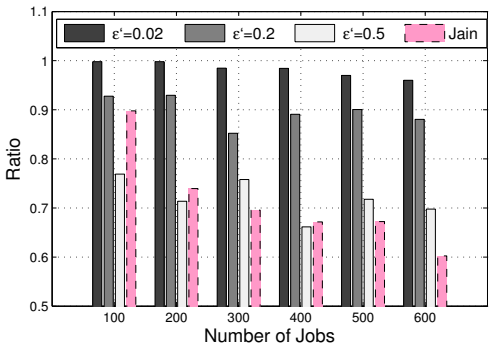

Figure 4.2: Performance ratio of $A_{offline}$, and Jain *et al.*'s algorithm [44].

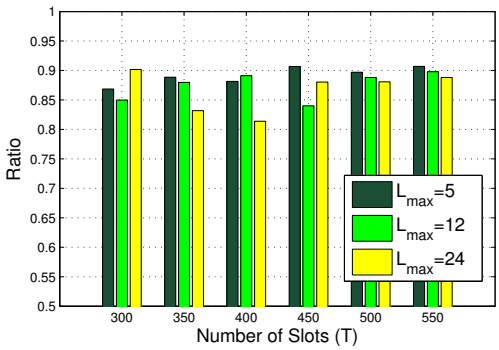

Figure 4.3: Performance ratio of $A_{offline}$ with different T and L_{max}.

Performance Ratio. We first examine the performance of our offline algorithm. The *performance ratio* is the ratio of the average objective value of ILP (4.1) generated by $A_{offline}$ to the optimal objective value of ILP (4.1). The average objective value is obtained by running lines 2-6 in Algorithm 6 20 times. We also implement Jain *et al.*'s offline algorithm [44], which proposes a greedy strategy to select winners, for comparison with $A_{offline}$. Fig. 4.2 shows that the performance ratio of $A_{offline}$ decreases slightly when we increase the total number of jobs. In addition, the ratio is inversely related to the input parameter ε' to Algorithm 6, as confirmed by the analysis in Theorem 7. $A_{offline}$ achieves a close-to-optimal performance with a small ε' (0.02) and has a better performance than Jain *et al.*'s algorithm even when ε' is relatively large (0.2). We next fix ε' to 0.2 and the number of jobs to 300, and vary the number of slots and the maximum length of subtasks. Fig. 4.3 illustrates that both T and $L_{\max}$ have relatively small impact on the performance of $A_{offline}$. This is because our offline solution is derived from the fractional solution rather than the input to the problem.

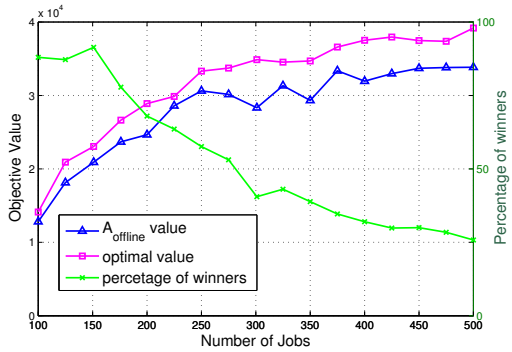

Figure 4.4: $A_{offline}$: objective value and percentage of winners.

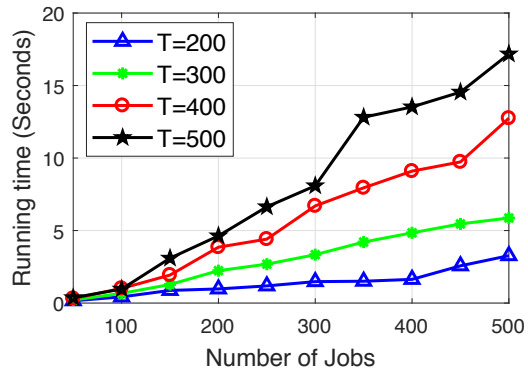

Figure 4.5: Running time of $A_{offline}$ under different I and T.

Objective Value, Winner Satisfaction. and Time Complexity. Fig. 4.4 compares the objective value produced by $A_{offline}$ to the optimal value. Again, there is just a small gap between these two values. The objective value grows with an increasing number of jobs because $A_{offline}$ can select more high-value jobs from a large set of jobs. The performance of $A_{offline}$ in terms of winner satisfaction, as measured by the percentage of winning jobs, is also demonstrated in Fig. 4.4. The percentage of winners drops when there is a large number of jobs. This is because the number of winners is relatively fixed and is limited by the resource capacity. Therefore, only a small percentage of jobs can be served from a large set of jobs. Next, we apply the `tic` and `toc` functions in MATLAB to measure the execution time of the main program without counting the initialization stage. We run 20 tests on a laptop computer (Intel Core i7-6700HQ/16GB RAM) and present the average result in Fig. 4.5. We can observe that the running time of $A_{offline}$ remains at a low level ($<$ 20 seconds) even when we input a large number of jobs and a long time span. It increases linearly with jobs and slots, and runs faster than the theoretical result indicated in Lemma 2.

4.6.2 Performance of A_{online}

Performance Ratio. The expected offline objective value is estimated by exactly solving ILP (4.1) 20 times under different realization of the bid arrival process. The *performance ratio* of A_{online} is the ratio of the average objective value produced by A_{online} (over different realizations of the bid

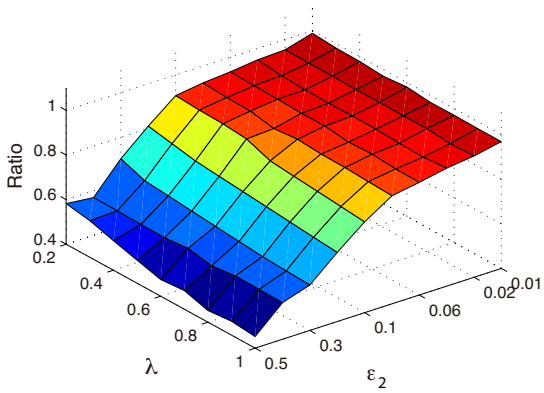

Figure 4.6: Performance ratio of A_{online} under different λ and ε_2.

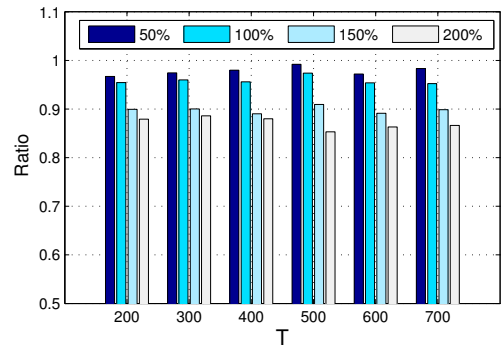

Figure 4.7: Performance ratio of A_{online} with different estimations of λ under different T.

arrival process) to the expected offline objective value. Fig. 4.6 shows that a better performance ratio comes with a smaller ε_2, while the arrival rate λ doesn't affect the ratio much. Comparing to the performance ratio of $A_{offline}$ in Fig. 4.2, we observe that both A_{online} and $A_{offline}$ can achieve a close-to-optimal performance and our online algorithm performs slightly worse than our offline algorithm since it doesn't have access to future job information. In the following figures, we fix the value of ε_2 to 0.02 and examine the impact of other parameters. We vary the total number of slots, use the estimated λ as input to A_{online} and plot the performance ratio in Fig. 4.7. We observe that the ratio remains relatively steady with the growth of T. Over-estimation causes a worse performance than under-estimation, as compared to the real λ (labelled by 100%). This is because A_{online} rejects more jobs with an over-estimated λ. The good news is that the ratio is still close to 0.9 even when we input an inaccurate λ.

We further compare our online algorithm with Zhou *et al.*'s online algorithm [84], which also conducts job admission based on the current resource prices. Their price is a function of U_k/L_k, where U_k (L_k) is the maximum (minimum) value per unit of type-k resource per unit of time. Fig. 4.8 and Fig. 4.9 show that A_{online} consistently outperforms Zhou *et al.*'s online algorithm over a wide range of U_k/L_k and number of slots (T). In Fig. 4.9, we set ε_2 to 0.2 and still observe the superiority of our online algorithm.

Objective Value and Winner Satisfaction. Next, we investigate the performance of A_{online}, in the

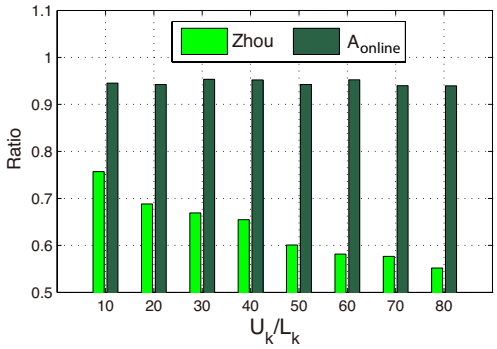

Figure 4.8: Comparison between A_{online} and Zhou *et al.*'s online algorithm [84] under different U_k/L_k.

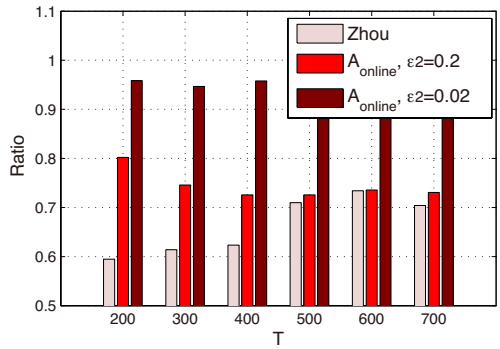

Figure 4.9: Comparison between A_{online} and Zhou under different T.

aspects of achieved objective value and winner satisfaction. In Fig. 5.9, there is an upward trend in the objective value with an increasing number of jobs. When ε_2 decreases, the solution output by A_{online} is closer to optimum, leading to a higher overall obtained value. Fig. 5.10 reflects that the percentage of winners also goes down with the increase of the number of jobs, similar to that of $A_{offline}$. Moreover, more jobs can be served when the number of subtasks in each job rises since there is larger selection space for each job's execution.

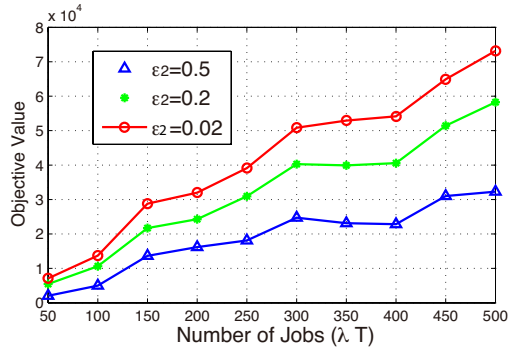

Figure 4.10: Objective value achieved by A_{online}.

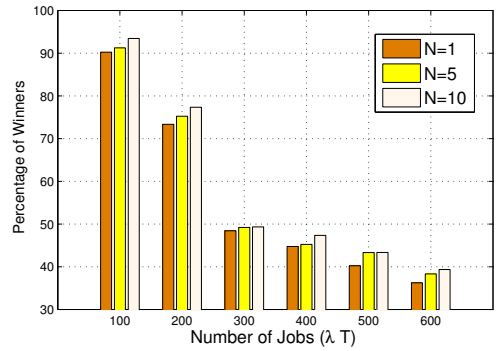

Figure 4.11: Percentage of winners in A_{online}.

Time Complexity and Feasibility. We test the running time of A_{online} under different input scales, and plot the result in Fig. 4.12. We can see that the worst case running time of A_{online} is shorter than 0.12 seconds, which is much smaller than that of $A_{offline}$. Moreover, its runtime slightly increases with the number of jobs and the number of time slots. The value of ε_2 determines its runtime.

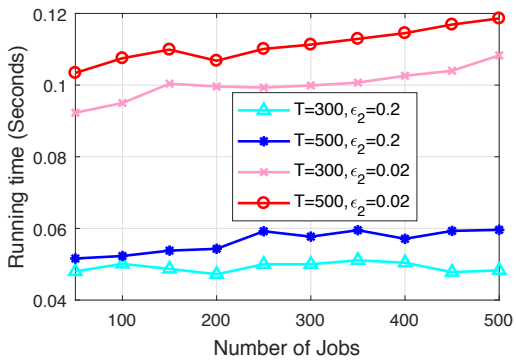

Figure 4.12: Running time of A_{online} under different I, T, and ε_2.

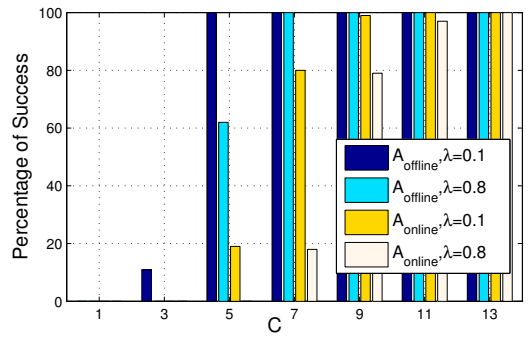

Figure 4.13: Feasibility test for $A_{offline}$ and A_{online} when $T = 500$.

This is because ε_2 is used to compute the number of times to solve dual LP (11). Finally, we run a feasibility test for $A_{offline}$ and A_{online}. In Theorem 7 and Lemma 17, we proved that with high probability, both $A_{offline}$ and A_{online} can produce feasible solutions. Therefore, we vary the value of C and the number of jobs (determined by λT). We run each algorithm 100 times, and count the number of successes, *i.e.,* the number of feasible solutions returned. As shown in Fig. 4.13, although we require C to be a large number in the theoretical proof, our simulation results show that both algorithms work well when $C > 10$. In addition, a larger number of jobs results in a lower success rate.

4.7 Summary

We presented both offline and online scheduling frameworks for cloud container services. We first apply a new LP formulation technique to handle scheduling constraints. In the offline algorithm design, we approximately compute the fractional solution by a separation oracle and round it to a feasible solution. In the online algorithm design, we apply the online learning technique to obtain the dual solution progressively. The dual solution acts as the resource price to facilitate the decision making. Our offline and online algorithms are expressive, computationally and economically efficient.

Chapter 5

An Efficient Online Placement Scheme for Cloud Container Clusters

5.1 Introduction

In this chapter, we extend the existing literature on virtual cluster provisioning, and propose an efficient online placement scheme such that: i) Container clusters (CCs) with different values arrive stochastically; each CC specifies its required containers and the traffic demand between neighbor containers; ii) the algorithm is computationally efficient and executes in polynomial time; iii) the aggregate value of deployed CCs is approximately maximized. Our detailed contributions are summarized below.

First, we formulate the offline optimization problem as an integer program (IP) with quadratic constraints that capture inter-container traffic flow. While polynomial in size, the quadratic IP is non-linear and admits no direct application of the classic primal-dual schema for algorithm design. We leverage the recent *compact-exponential* optimization framework [84] to encode each valid placement scheme in a variable, and reformulate the original IP into a *compact-exponential Integer Linear Program (ILP)*, which contains only conventional packing-type constraints, but at the cost of involving an exponential number of variables. Solving the compact exponential ILP directly is still infeasible in practice, when complete knowledge over the entire system lifespan is not available. We instead first relax the resource capacity constraints that impose inter-CC coupling, and focus on a one-shot problem to determine the optimal placement of the current CC. We then design an online algorithm framework that simultaneously works on the compact-exponential ILP and its dual LP, invoking the one-shot algorithm as a subroutine, towards computing efficient placement based on values of dual variables.

Second, we reformulate the one-shot CC placement problem into an integer quadratic program (IQP), to minimize the placement cost of a given CC. We first consider a simplified scenario of a single type of computational resource. The IQP has an objective function of degree 2, and is proven NP-hard to solve. We apply an *exhaustive sampling* technique [17] based on a random-sampling process to reduce its degree from 2 to 1, at the cost of losing some accuracy. The degree-reduced problem becomes a general assignment problem with extra constraints. We solve this problem to optimum, and apply the *ST rounding* technique [63] to round the fractional solution to an integral solution. More specifically, we construct a bipartite graph based on the fractional solution, and output the minimum-cost integer matching in this graph. Theoretical analysis shows that our algorithm achieves a small approximation ratio. We then further consider the general scenario, and propose a heuristic algorithm to provide good solutions with low computational complexity.

Third, we proceed to consider resource capacity constraints, and design an online algorithm framework that utilizes the one-shot algorithm to determine each CC's placement upon its arrival, without relying on future information. We apply the primal-dual technique to the compact-exponential ILP and its dual LP, and interpret dual variables as unit resource prices at different times. Upon receiving a CC request, given current resource prices, the one-shot algorithm computes an α-approximate placement scheme with an estimated cost. We divide the estimated cost by α to obtain a lower bound of the optimal cost, and compare the CC's value with it. If the value is higher, the CC is deployed and dual variables are updated; otherwise the CC request is discarded. We conduct theoretical analysis on the competitive ratio and prove its upper bound. The effectiveness of our one-shot and online algorithms are evaluated through trace-driven simulation studies.

In the rest of this Chapter, we review related work in Ch. 5.2 and describe the system model in Ch. 5.3. We study the one-shot CC placement problem in Ch. 5.4 and propose an online placement algorithm on Ch. 5.5. Simulation studies are presented in Ch. 5.6. Ch. 5.7 concludes the chapter.

5.2 Related Work

Early studies on cloud resource management have focused on VM provisioning, in both offline and online settings. Zhang *et al.* [78] propose a randomized algorithm based on a decomposition technique for dynamic cloud resource provisioning, achieving a small approximation ratio. Shi *et al.* [62] present the first online combinatorial auction in the cloud computing paradigm. Zhang *et al.* [82] design online multi-resource allocation algorithms to schedule cloud jobs with deadlines. The above literature focuses on the deployment of separate VMs, without considering inter-VM traffic in a virtual cluster.

Cloud container cluster provisioning belongs to the category of virtual cluster (VC) provisioning (*a.k.a.* virtual network embedding/mapping). Along this direction, Chowdhury *et al.* [28] propose virtual network embedding algorithms that efficiently map virtual nodes and virtual links onto substrate network resources. Li *et al.* [49] address the VM placement problem, by considering both the traffic cost and the physical machine utilization cost. Yu *et al.* [74] study the survivable VC embedding problem, which jointly optimizes primary and backup embeddings of VCs, with the goal of minimizing VM consumption. Dai *et al.* [30] design algorithms for the minimum energy virtual cluster embedding problem. They provide no proven guarantee on the approximation ratio. Different from the above literature, our one-shot CC placement problem has a different optimization objective. We target total cost minimization with a natural IQP formulation, requiring different solution techniques. We rigorously prove that our proposed algorithm can achieve a small approximation ratio. In addition, the above literature considers only one-time/offline scenario, while we further propose an efficient online CC placement scheme to handle dynamically arriving requests for CCs.

Towards online VC deployment, Evan *et al.* [32] study the online multi-commodity flow routing problem. They focus on link capacity constraints but ignore node capacity constraints. Grandl *et al.* [38] propose a multi-resource cluster scheduler that assigns tasks to machines. The communication demand between different tasks is not modelled by them. Shi *et al.* [61] investigate online

mechanism design to place inter-connected VMs in a geo-distributed IaaS cloud, taking both computational resources and communication resources into consideration. Their subproblem for each job's placement is trivial, since they specify several VM placement schemes for each job, while our subproblem is an NP-hard problem that computes the best placement for each CC. Our work is also the first to design an online primal-dual algorithm for CC placement, with proven performance guarantee.

The compact-exponential optimization framework was first applied by Zhou *et al.* [84]. They consider the scheduling of computing jobs that require separate VMs, while this chapter focuses on the placement of correlated containers in the form of container clusters. This chapter further advances the compact-exponential framework to handle nonlinear constraints and NP-hard subproblems. Our subproblem, namely the one-shot CC placement problem, is a special case of the quadratic assignment problem (see [51] for a detailed survey). We design a rounding algorithm that combines exhaustive sampling [17] and ST rounding [63] techniques for effective solutions. The online primal-dual method is a known powerful algorithmic technique for many NP-hard problems, such as the knapsack problem and the general packing problem [24]. However, our online optimization problem does not fall into such known categories. We propose a primal-dual online framework to solve our problem, and provide a new price function to update dual variables, which is the key towards achieving a good competitive ratio.

5.3 System Model

We consider a cloud service provider who owns a pool of resources residing in S zones, where a *zone* may correspond to one server or a cluster of servers, or a data center. Let $[X]$ denote the integer set $\{1,2,\ldots,X\}$. There are K types of computation resources, as exemplified by CPU, RAM and disk. Each zone $s \in [S]$ has C_{ks} units of type-k resource. Zones are interconnected by broadband links. Active optical cables (AOC) and unshielded twisted pair (UTP) cables are often used for short links that connect zones in the same data center, while multi-mode or single-mode

fibers are used to connect zones which correspond to different data centers [4]. Let $\mathbb{E}$ be the set of links, and let D_{s_1,s_2} denote the bandwidth capacity of link $(s_1,s_2) \in \mathbb{E}$ that connects zones s_1 and s_2.

Over a large time span $1,2,\ldots,T$, I CC requests arrive stochastically to the system. Multiple requests can arrive simultaneously, and would be ordered randomly. Request i arrives at time t_i, requiring a CC from t_i^- to t_i^+. Each CC consists of a set of tailor-made containers. Let $\mathscr{V}_i$ and V_i denote the set of containers and the number of containers in request i's CC, respectively. A container $v \in \mathscr{V}_i$ consumes a^i_{vk} amount of type-k resource, $\forall k \in [K]$. Let $\Delta^i_{v_1,v_2}$ denote the bandwidth consumption for flow transfer from v_1 to v_2 in request i's CC, when v_1 and v_2 reside distinct zones. A value b_i is obtained if request i's CC is deployed. In summary, request i can be expressed as:

$$\Phi_i = \{b_i, V_i, \mathscr{V}_i, \{a^i_{vk}\}_{v\in\mathscr{V}_i,k\in[K]}, \{\Delta^i_{v_1,v_2}\}_{v_1,v_2\in\mathscr{V}_i}\}.$$

Upon each request's arrival, the service provider immediately determines whether to serve it, and if so, how to place its CC. Decision variables for request i include: i) $x_i \in \{0,1\}$, indicating whether request i is accepted (1) or not (0). ii) $y^i_{vs}, \forall v \in \mathscr{V}_i, \forall s \in [S]$, encoding the placement scheme of request i's CC, where $y^i_{vs} = 1$ if zone s is selected to host container v and 0 otherwise. The service provider in practice wishes to reserve resources for different CC requests, and limits a single CC to occupy at most B_{ks} units of type-k resource in zone s. Such resource consumption bound is also customary in the cloud resource allocation literature [53] [82]. Fig. 5.1 shows a placement scheme for request 1. Our objective is to maximize the total valuation obtained from all CCs, subject to resource capacity constraints. The optimization problem can be formulated into the following integer program (IP):

$$\text{maximize} \quad \sum_{i\in[I]} b_i x_i \tag{5.1}$$

subject to:

$$x_i \geq \sum_{s\in[S]} y^i_{vs}, \forall i \in [I], \forall v \in \mathscr{V}_i, \tag{5.1a}$$

$$\sum_{\substack{i\in[I]:\\ t_i^- \leq t \leq t_i^+}} \sum_{v\in\mathscr{V}_i} a^i_{vk} y^i_{vs} \leq C_{ks}, \forall k \in [K], \forall s \in [S], \forall t \in [T], \tag{5.1b}$$

$$\sum_{s\in[S]} y^i_{vs} \leq 1, \forall i \in [I], \forall v \in \mathscr{V}_i, \tag{5.1c}$$

$$\sum_{v\in\mathscr{V}_i} a^i_{vk} y^i_{vs} \leq B_{ks}, \forall i \in [I], \forall k \in [K], \forall s \in [S], \forall t \in [t_i^-, t_i^+] \tag{5.1d}$$

$$\sum_{\substack{i\in[I]:\\ t_i^- \leq t \leq t_i^+}} \sum_{v_1,v_2\in\mathscr{V}_i} \Delta^i_{v_1,v_2} y^i_{v_1,s_1} y^i_{v_2,s_2} \leq D_{s_1,s_2},$$

$$\forall (s_1,s_2) \in \mathbb{E}, \forall t \in [T], \tag{5.1e}$$

$$x_i, y^i_{vs} \in \{0,1\}, \forall i \in [I], \forall v \in \mathscr{V}_i, \forall s \in [S]. \tag{5.1f}$$

Constraints (5.1a) ensures that request i's CC is deployed only when it is accepted, since request i's container v is placed to a zone s only when $x_i = 1$. Constraint (5.1b) guarantees that at any time, allocated resources at a zone do not exceed its capacity. Constraint (5.1c) indicates that a container in a CC request resides in at most one zone. Constraint (5.1d) enforces the upper-bound of each CC's resource occupation at a zone s. Link capacity constraints are modelled by (5.1e).

Even in the offline setting, with complete knowledge given, the polynomial-sized IP (5.1) is NP-hard to solve. To verify, consider a special case of IP (5.1) where each CC consists of one container, $T = 1$ and $B_{ks} = D_{s_1,s_2} = +\infty$. Then the classic multidimensional knapsack problem, which is known to be NP-hard, is equivalent to the special case of IP (5.1). The challenge further escalates when we consider quadratic constraints (5.1e). To address these challenges, we resort to the compact-exponential technique [84], which can reformulate IP (5.1) into an equivalent ILP with packing structure, at the price of involving an exponential number of variables:

$$\text{maximize} \sum_{i\in[I]} \sum_{l\in\zeta_i} b_{il} x_{il} \tag{5.2}$$

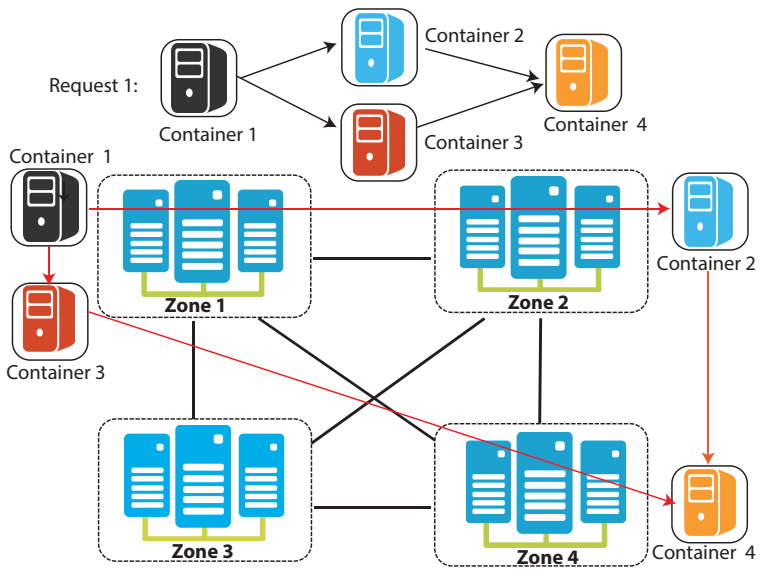

Figure 5.1: Container cluster placement: an example.

subject to:

$$\sum_{\substack{i\in[I]:\\ t_i^- \le t \le t_i^+}} \sum_{l\in\zeta_i} f_{m,t}^{il} x_{il} \le C_m, \forall m \in \mathscr{M}, \forall t \in [T], \tag{5.2a}$$

$$\sum_{l\in\zeta_i} x_{il} \le 1, \forall i \in [I], \tag{5.2b}$$

$$x_{il} \in \{0,1\}, \forall i \in [I], \forall l \in \zeta_i. \tag{5.2c}$$

In the above compact-exponential ILP, ζ_i is the set of feasible placement schemes for request i. A *feasible* scheme is a vector $l = \{y_{vs}^i\}$ that satisfies (5.1c) and (5.1d). Variable $x_{il} \in \{0,1\}$ indicates whether request i's scheme l is accepted (1) or not (0). We regard each computation resource at each zone and the bandwidth at each link as different resources. Consequently, the total number of resource types is $KS + |\mathbb{E}|$. Let $\mathscr{M}$ be the set of resource types and C_m be the capacity of type-m resource, $\forall m \in \mathscr{M}$. $f_{m,t}^{il}$ denotes the total type-m resource consumption of request i's scheme l at time t. For example, if m corresponds to type-k resource at zone s, $f_{m,t}^{il} = \sum_{v\in\mathscr{V}_i} a_{vk}^i y_{vs}^i, \forall t \in [t_i^-, t_i^+]$. Constraint (5.2a) is equivalent to (5.1b) and (5.1e). Constraint (5.2b) ensures that each CC is placed according to at most one scheme.

We relax $x_{il} \in \{0,1\}$ to $x_{il} \ge 0$, and introduce dual variables $p_{m,t}$ and u_i to constraints (5.2a)

and (5.2b). The dual of the relaxation of program (5.2) is:

$$\text{minimize} \sum_{t\in[T]} \sum_{m\in\mathscr{M}} C_m p_{m,t} + \sum_{i\in[I]} u_i \tag{5.3}$$

subject to:

$$u_i \geq b_{il} - \sum_{m\in\mathscr{M}} \sum_{t\in[t_i^-,t_i^+]} f_{m,t}^{il} p_{m,t}, \forall i \in [I], \forall l \in \zeta_i, \tag{5.3a}$$

$$p_{m,t}, u_i \geq 0, \forall i \in [I], \forall m \in \mathscr{M}, \forall t \in [T]. \tag{5.3b}$$

IP (5.1) and ILP (5.2) have the same optimal objective value. To solve ILP (5.2), complete knowledge over the entire system lifespan is required. However, our algorithm needs to work in an online fashion, making on-spot decisions without relying on knowledge of future request arrivals. To this end, we leverage the primal-dual technique that determines the primal solution based on dual variables. We interpret dual variable $p_{m,t}$ as the unit price of type-m resource at time t. Upon the arrival of a request, we compute its CC's placement scheme based on current resource prices. We first focus on a one-shot CC placement problem, which relaxes resource capacity constraints (5.2a) that impose temporal correlation in online decision making; we design an efficient algorithm to determine a CC's placement scheme with the goal of cost minimization. We then propose an online algorithm framework that employs the one-shot optimization as a building block to make on-spot decisions upon CC request arrivals.

We make two assumptions in this Chapter. First, we assume that a CC's valuation is proportional to its resource consumption in each time slot $f_{m,t}^{il}$ (where $f_{m,t}^{il} > 0$): $1 \leq \frac{b_i}{f_{m,t}^{il}} \leq U, \forall i,l,m,t$, where U is a constant. Second, we assume that the ratio between a scheme's resource consumption at each time slot and the resource capacity is bounded: $\frac{f_{m,t}^{il}}{C_m} \leq \frac{1}{\log \lambda}, \forall i,l,m,t$, where $\lambda = 2(\alpha U + 1)$ and α represents the approximation ratio of the one-shot CC placement algorithm. This assumption implies that the resource demand of each container is small compared to the capacity of each zone. Here λ (related to U) is an important parameter and will be used in our online algorithm design. Notations are summarized in Table 5.1 for ease of reference.

Table 5.1: Summary of Notation in Chapter 5

I	# of CC requests	$[X]$	integer set $\{1,\ldots,X\}$	S	# of zones	$\mathbb{E}$	set of links
b_i	the value of request i' CC	K	# of types of computational resource				
T	# of time slots	C_{ks}	capacity of type-k resource at zone s				
$t_i^-(t_i^+)$	start (end) time of request i	D_{s_1,s_2}	bandwidth capacity of link (s_1,s_2)				
$\mathscr{V}_i(V_i)$	set(number) of containers in request i's CC						
a_{vk}^i	amount of type-k resource consumed by $v \in \mathscr{V}_i$						
Δ_{v_1,v_2}^i	traffic from v_1 to v_2 in request i's CC						
B_{ks}	upper bound of type-k resource consumption						
x_i	request i is accepted (1) or not (0)						
y_{vs}^i	container v is assigned to zone s or not in i's CC						
x_{il}	request i's scheme l is accepted (1) or not (0)						
$f_{m,t}^{il}$	demand of type-m resource at t by i's scheme l						
$p_{m,t}$	cost of unit type-m resource at t						
λ	$2(\alpha U+1)$, where $1 \leq \frac{b_i}{f_{m,t}^{il}} \leq U, \forall i,l,m,t$						

5.4 Approximation Algorithm Design for Container Cluster Placement

In this section, we first formulate the one-shot CC placement problem in Ch. 5.4.1. A rounding algorithm and a heuristic algorithm are then designed and analyzed in Ch. 5.4.2 and Ch. 5.4.3, respectively.

5.4.1 Cost Minimization Problem

We include the same constraints (5.1c) and (5.1d) related to the current CC request i from IP (5.1), and exclude resource capacity constraints (5.1b) and (5.1e). Given current resource prices (to be decided in Ch. 5.5), we compute the computation cost P_{vs}^i, *i.e.,* the cost of placing container v in zone s, and the communication cost P_{v_1,v_2,s_1,s_2}^i, *i.e.,* the cost of sending traffic from v_1 to v_2 through link (s_1,s_2). The cost minimization problem has the following natural IQP formulation:

$$\text{minimize} \sum_{v\in\mathscr{V}_i}\sum_{s\in[S]} P_{vs}^i y_{vs}^i + \sum_{\substack{v_1,v_2\in\mathscr{V}_i,\\(s_1,s_2)\in\mathbb{E}}} P_{v_1,v_2,s_1,s_2}^i y_{v_1,s_1}^i y_{v_2,s_2}^i \tag{5.4}$$

subject to:

$$\sum_{v \in \mathscr{V}_i} a^i_{vk} y^i_{vs} \leq B_{ks}, \forall k \in [K], \forall s \in [S], \tag{5.4a}$$

$$\sum_{s \in [S]} y^i_{vs} = 1, \forall v \in \mathscr{V}_i, \tag{5.4b}$$

$$y^i_{vs} \in \{0,1\}, \forall v \in \mathscr{V}_i, \forall s \in [S]. \tag{5.4c}$$

If we set $a^i_{vk} = B_{ks} = 1, \forall v, k, s$, IQP (5.4) degrades to the minimum quadratic assignment problem [51], which has been proven NP-hard, and does not have any known constant-factor approximation algorithm in polynomial time (assuming P $\neq$ NP). When the number of zones is small ($S < 5$), we can compute the optimal solution by enumerating all options. However, S can be a large number in practice. For example, the Amazon cloud infrastructure is currently comprised of 42 availability zones [13]. It is essential to compute a good solution efficiently, in a short time. To solve the IQP, we first simplify the model by considering a single type of computation resource, and present a rounding algorithm that approximately solves the problem with worst-case performance guarantee. We later propose a heuristic algorithm that can solve the general version, without theoretical performance guarantee.

5.4.2 A Rounding Algorithm with Performance Guarantee

Given a sole type of computation resource, let a^i_v and B_s represent a^i_{vk} and B_{ks}, respectively. The *degree* of an integer program is d if its objective function is a degree-d polynomial function. IQP (5.4) is a degree-2 integer program. We first apply *exhaustive sampling* [17] to reduce its degree from 2 to 1. We convert IQP (5.4) to an ILP, with some loss of accuracy. The ILP is a generalized assignment problem with side constraints. We solve its LP relaxation exactly, and then round the fractional solution to an approximate integral solution through another technique of *ST rounding* [63].

More specifically, let ${y^i_{vs}}^*$ be the optimal solution to IQP (5.4), and let

$${F^i_{v_1,s_1}}^* = \sum_{v_2 \in \mathscr{V}_i} \sum_{s_2 \in [S]} P^i_{v_1,v_2,s_1,s_2} {y^i_{v_2,s_2}}^*,$$

we can reformulate IQP (5.4) to the following ILP:

$$\text{minimize} \quad \sum_{v \in \mathscr{V}_i} \sum_{s \in [S]} (P^i_{vs} + {F^i_{vs}}^*) y^i_{vs} \tag{5.5}$$

subject to:

$$\text{Constraints} \ \ (5.4a - 5.4c)$$

$$\sum_{v_2 \in \mathscr{V}_i} \sum_{s_2 \in [S]} P^i_{v_1,v_2,s_1,s_2} y^i_{v_2,s_2} = {F^i_{v_1,s_1}}^*, \forall v_1 \in \mathscr{V}_i, \forall s_1 \in [S]. \tag{5.5d}$$

However, it is difficult to obtain the exact value of ${F^i_{v_1,s_1}}^*$. We strive to compute reasonable guesses by exhaustively listing all placement schemes of a random sample. For ease of presentation, we normalize $P^i_{v_1,v_2,s_1,s_2}$ and P^i_{vs}, so that $\max\left\{\max_{v_1,v_2,s_1,s_2}\{P^i_{v_1,v_2,s_1,s_2}\}, \max_{v,s}\{P^i_{vs}\}\right\} = 1$. The sampling procedure is as follows:

i) We first pick a random sample $\mathscr{W}$ of $n = O(\log V_i/\varepsilon^2)$ containers from $\mathscr{V}_i$. Let $\mathscr{W} = \{v_{j_1}, \ldots, v_{j_n}\}$. Exhaustively go through each of the S^n ways of placing containers in $\mathscr{W}$. For each placement that satisfies constraints (5.4a) and (5.4b), we compute an estimate $F^i_{v_1,s_1}$ of ${F^i_{v_1,s_1}}^*$ by setting

$$F^i_{v_1,s_1} = \frac{V_i}{n} \sum_{v_2 \in \mathscr{W}} \sum_{s_2 \in [S]} P^i_{v_1,v_2,s_1,s_2} y^i_{v_2,s_2},$$

where $y^i_{v_2,s_2} = 1$ if container v_2 is allocated to zone s_2. Note that we try all possible placements of containers in $\mathscr{W}$. Therefore, the "correct" placement in which $y^i_{v_2,s_2} = {y^i_{v_2,s_2}}^*, \forall v_2 \in \mathscr{W}, s_2 \in [S]$ is ensured to be tried. We call the estimate corresponding to this assignment *the special estimate*, denoted as ${F^i_{v_1,s_1}}^s$. We will show that ${F^i_{v_1,s_1}}^s$ satisfies $|{F^i_{v_1,s_1}}^* - {F^i_{v_1,s_1}}^s| \leq \varepsilon V_i$ in Lemma 24.

ii) Next, for each estimate $F^i_{v_1,s_1}$ of ${F^i_{v_1,s_1}}^*$, we consider the following LP:

$$\text{minimize} \quad \sum_{v \in \mathscr{V}_i} \sum_{s \in [S]} (P^i_{vs} + F^i_{vs}) y^i_{vs} \tag{5.6}$$

subject to:

Constraints $(5.4a - 5.4b)$

$$\sum_{v_2 \in \mathscr{V}_i} \sum_{s_2 \in [S]} P^i_{v_1,v_2,s_1,s_2} y^i_{v_2,s_2} \leq F^i_{v_1,s_1} + \varepsilon V_i, \forall v_1 \in \mathscr{V}_i, \forall s_1 \in [S], \tag{5.6c}$$

$$\sum_{v_2 \in \mathscr{V}_i} \sum_{s_2 \in [S]} P^i_{v_1,v_2,s_1,s_2} y^i_{v_2,s_2} \geq F^i_{v_1,s_1} - \varepsilon V_i, \forall v_1 \in \mathscr{V}_i, \forall s_1 \in [S], \tag{5.6d}$$

$$y^i_{vs} \geq 0, \forall v \in \mathscr{V}_i, \forall s \in [S]. \tag{5.6e}$$

Let $\overline{y^i_{vs}}$ and $\overline{\eta}$ be the optimal solution and the optimal objective value of LP (5.6), respectively. We round the fractional solution $\overline{y^i_{vs}}$ to an approximate integral solution ${y^i_{vs}}'$ using an algorithm A_{round}. The approximate integral solution satisfies constraint (5.4b) and slightly violates (5.4a) by allowing each CC to occupy at most twice its resource usage bound at each zone.

iii) Because there are at most S^n estimates, we solve LP (5.6) as above for each $F^i_{v_1,s_1}$. Among the solutions, we output the one whose corresponding placement minimizes the objective of IQP (5.4). We summarize our algorithm in A_{sub1}.

We next describe the details of A_{round}, which applies ST rounding [63] to round the fractional solution to an integral solution. In preparation, we first construct a weighted bipartite graph $\mathscr{B} = (V', S', E')$ based on $\overline{y^i_{vs}}$, as follows:

i) One side of the bipartite graph consists of container nodes: $V' = \{v_v : v = 1, \ldots, V_i\}$, and the other side includes zone nodes: $S' = \{s_{s\theta} : s = 1, \ldots, S, \theta = 1, \ldots, \Theta_s\}$, where $\Theta_s = \lceil \sum_{v \in \mathscr{V}_i} \overline{y^i_{vs}} \rceil$. We assign Θ_s nodes for zone s.

ii) For zone s, sort the containers placed to it by non-increasing resource demand a^i_v. For notation simplicity, we assume that $a^i_1 \geq a^i_2 \ldots, \geq a^i_{V_i}$.

iii) For zone s, consider nodes $s_{s\theta}, \theta = 1, \ldots, \Theta_s$ as bins of capacity 1, and consider $\overline{y^i_{vs}}, v \in \mathscr{V}_i$ as pieces of containers placed into these bins. With respect to the non-increasing order of a^i_v, we pack pieces into node s_{s1} one by one, until placing piece v results in bin overflow (exceeding 1). We then place a fraction of v to saturate the capacity of node s_{s1}, and place the remaining fraction of v to node s_{s2}. We carry on this process until all pieces are placed into bins.

iv) If there is a positive fraction of container v packed into node $s_{s\theta}$, edge $(v_v, s_{s\theta})$ is added to E'. We define a vector $y'(v_v, s_{s\theta})$ on edge $(v_v, s_{s\theta})$, which equals the fraction of v packed into $s_{s\theta}$. The cost of this edge, $w(v_v, s_{s\theta})$, is set to $P^i_{vs} + F^i_{vs}$.

Vector y' is a fractional matching in bipartite graph $\mathscr{B}$ that exactly matches all container nodes. The cost of the fractional matching is $\overline{\eta}$. By matching theory [52], there exists an integral matching with cost at most $\overline{\eta}$ that exactly matches all container nodes. We continue to compute such an integral matching.

Algorithm 9 An Approximation Algorithm A_{sub1}.

Input: $\Phi_i, \{P^i_{vs}\}, \{P^i_{v_1,v_2,s_1,s_2}\}, \{a^i_v\}, \varepsilon$

Define $n = O(\log V_i/\varepsilon^2)$;
Pick a random subset $\mathscr{W}$ of n containers from $\mathscr{V}_i$; Let $\mathscr{W} = \{v_{j_1}, \ldots, v_{j_n}\}$;
for each placement of containers in $\mathscr{W}$ **do**
    Update the corresponding y^i_{vs};
    Define $F^i_{v_1,s_1} = \frac{V_i}{n} \sum_{v_2 \in \mathscr{W}} \sum_{s_2 \in [S]} P^i_{v_1,v_2,s_1,s_2} y^i_{v_2,s_2}, \forall v_1 \in \mathscr{V}_i, s_1 \in [S]$;
    Solve LP (5.6) exactly. Let $\overline{y^i_{vs}}$ be the optimal solution;
    $\{{y^i_{vs}}'\} = A_{round}(\{\overline{y^i_{vs}}\}, \{P^i_{vs} + F^i_{vs}\})$;
    $\Lambda_j = \sum_{v \in \mathscr{V}_i} \sum_{s \in [S]} P^i_{vs} {y^i_{vs}}' + \sum_{v_1,v_2 \in \mathscr{V}_i} \sum_{(s_1,s_2) \in \mathbb{E}} P^i_{v_1,v_2,s_1,s_2} {y^i_{v_1,s_1}}' {y^i_{v_2,s_2}}'$;
end for
$cost_i = \min_j \Lambda_j$; Save the placement $\{y^i_{vs}\}$ which leads to the minimum Λ_j to l^*;
Return $cost_i, l^*$.

Algorithm 10 A Rounding Algorithm A_{round}

Input: $\{\overline{y^i_{vs}}\}, \{P^i_{vs} + F^i_{vs}\}$;

Build a bipartite graph $\mathscr{B} = (V', S', E')$ based on $\overline{y^i_{vs}}$;
Find the minimum-cost integer matching that exactly matches all container nodes in $\mathscr{B}$;
Update ${y^i_{vs}}' = 1$ if node v is mapped to node s;
Return $\{{y^i_{vs}}'\}$.

Theoretical Analysis

i) Feasibility and Running Time.

Theorem 10. The integral solution y^i_{vs} returned by A_{sub1} satisfies constraint (5.4b) and slightly violates constraint (5.4a) by allowing $\sum_{v \in \mathscr{V}_i} a^i_v y^i_{vs} \leq 2B_s, \forall s \in [S]$.

Proof: A_{round} generates an integral matching that exactly matches all container nodes. As a result, constraint (5.4b) is satisfied. We next examine constraint (5.4a). For each zone node $s_{s\theta}$, let $a_{s\theta}^{\max}$ denote the maximum of a_v^i corresponding to edges $(v_v, s_{s\theta}), \forall v_v$, and let $a_{s\theta}^{\min}$ denote the corresponding minimum. We have $a_{s\theta}^{\min} \geq a_{s,\theta+1}^{\max}, \theta = 1, \ldots, \Theta_s - 1$. Then the total resource consumption of containers assigned to zone s by any integral matching in $\mathscr{B}$ is at most $\sum_{\theta=1}^{\Theta_s} a_{s\theta}^{\max}$. Clearly, $a_{s1}^{\max} \leq B_s$.

$$\sum_{\theta=2}^{\Theta_s} a_{s\theta}^{\max} \leq \sum_{\theta=1}^{\Theta_s-1} a_{s\theta}^{\min} \leq \sum_{\theta=1}^{\Theta_s-1} \sum_{v:(v_v,s_{s\theta})\in E'} a_v^i y'(v_v, s_{s\theta})$$
$$\leq \sum_{\theta=1}^{\Theta_s} \sum_{v:(v_v,s_{s\theta})\in E'} a_v^i y'(v_v, s_{s\theta}) = \sum_{v\in\mathscr{V}_i} a_v^i \overline{y_{vs}^i} \leq B_s,$$

which proves the theorem. □

Theorem 11. A_{sub1} is a polynomial time algorithm, with time complexity $O(S^{n+1}V_i^2 \log V_i)$.

Proof: Lines 1-2 take $O(n)$ steps to initialize set $\mathscr{W}$. The `for` loop in lines 3-9 iterates at most S^n times. In each iteration, lines 4-5 can be accomplished in $O(SV_i)$ steps. The complexity of the ST rounding algorithm in lines 6-7 is $O(SV_i^2 \log V_i)$ [63]. Line 8 computes the objective value in $O(SV_i)$ steps. Thus, the complexity of the `for` loop is $O(S^{n+1}V_i^2 \log V_i)$. Lines 10-11 return the output in $O(S^n)$ time. In summary, the time complexity of A_{sub2} is $O(S^{n+1}V_i^2 \log V_i)$, which is polynomial to the input size. □

ii) Approximation Ratio.

The *approximation ratio* is the upper-bound ratio of the objective value achieved by A_{sub1} to the optimal objective value of IQP (5.4).

Lemma 23. (**Chernoff bound**, Lemma 24 in [16]) Let $X_1, \ldots, X_k$ be independent random variables such that $0 \leq X_i \leq 1$. Let $X = \sum_{i=1}^k X_i$ and $\mu = E(X)$,

$$Pr[|X - \mu|] \geq \sigma] \leq 2e^{-2\sigma^2/k}.$$

Lemma 24. Assume $n = q \log V_i / \varepsilon^2$, the special estimate ${F_{v_1,s_1}^i}^s$, with probability at least $1 - \frac{2}{V_i^{2q}}$, is in the range of $[{F_{v_1,s_1}^i}^* - \varepsilon V_i, {F_{v_1,s_1}^i}^* + \varepsilon V_i]$.

Proof: We define n random variables $Y_1,\ldots,Y_n$ and let $Y_j=\sum_{s_2\in[S]}P_{v_1,v_2,s_1,s_2}{y^i_{v_j,s_2}}^*, \forall v_j\in\mathscr{W}$. Note that $0\le Y_j\le 1, j=1,\ldots,n$. Then $Y=\sum_{j=1}^n Y_j=\frac{{F^i_{v_1,s_1}}^s n}{V_i}$ and $E[Y]=\frac{n}{V_i}{F^i_{v_1,s_1}}^*$. By Lemma 23, we have

$$Pr[|{F^i_{v_1,s_1}}^s-{F^i_{v_1,s_1}}^*|\ge\varepsilon V_i]=Pr[\frac{n}{V_i}|{F^i_{v_1,s_1}}^s-{F^i_{v_1,s_1}}^*|\ge\varepsilon n]$$
$$=Pr[|Y-E[Y]|\ge\varepsilon n]\le 2e^{-2\varepsilon^2 n}=2e^{-2\varepsilon^2 q\log V_i/\varepsilon^2}=\frac{2}{V_i^{2q}}.$$

Thus, $Pr[|{F^i_{v_1,s_1}}^s-{F^i_{v_1,s_1}}^*|\le\varepsilon V_i]\ge 1-\frac{2}{V_i^{2q}}$. □

Lemma 25. Upon termination, A_{sub1} outputs an integral solution y^i_{vs} that satisfies

$$\Lambda(y^i_{vs})\le\Lambda^*+(1+\varepsilon)V_i^2,$$

where $0\le\varepsilon\le 1$, $\Lambda(y^i_{vs})$ is the objective value of IQP (5.4) produced by y^i_{is} and Λ^* is the optimal objective value of IQP (5.4).

Proof: Recall that ${y^i_{vs}}^*$ is the optimal solution to IQP (5.4). Lemma 24 implies that with high probability, ${y^i_{vs}}^*$ is a feasible solution to LP (5.6) when we input the special estimate ${F^i_{v_1,s_1}}^s$. Because $\overline{y^i_{vs}}$ is the optimal solution to LP (5.6), the integral solution ${y^i_{vs}}'$, which is rounded from $\overline{y^i_{vs}}$, satisfies:

$$\sum_{v\in\mathscr{V}_i}\sum_{s\in[S]}(P^i_{vs}+{F^i_{vs}}^s){y^i_{vs}}'=\sum_{v\in\mathscr{V}_i}\sum_{s\in[S]}(P^i_{vs}+{F^i_{vs}}^s)\overline{y^i_{vs}}\le\sum_{v\in\mathscr{V}_i}\sum_{s\in[S]}(P^i_{vs}+{F^i_{vs}}^s){y^i_{vs}}^*$$
$$\le\sum_{v\in\mathscr{V}_i}\sum_{s\in[S]}P^i_{vs}{y^i_{vs}}^*+\sum_{v_1\in\mathscr{V}_i}\sum_{s_1\in[S]}({F^i_{v_1,s_1}}^*+\varepsilon V_i){y^i_{v_1,s_1}}^*=\Lambda^*+\varepsilon V_i(\sum_{v\in\mathscr{V}_i}\sum_{s\in[S]}{y^i_{vs}}^*)=\Lambda^*+\varepsilon V_i^2.$$

Because $0\le P^i_{v_1,v_2,s_1,s_2}\le 1$, we can obtain $\sum_{v_2\in\mathscr{V}_i}\sum_{s_2\in[S]}P^i_{v_1,v_2,s_1,s_2}{y^i_{v_2,s_2}}'\le{F^i_{vs}}^s+V_i$. As a result, $\Lambda({y^i_{vs}}')$, the objective value of IQP (5.4) achieved by ${y^i_{vs}}'$, is at most $\sum_{v\in\mathscr{V}_i}\sum_{s\in[S]}(P^i_{vs}+{F^i_{vs}}^s){y^i_{vs}}'+V_i^2$.

Among different rounded integral solutions, the output y^i_{is} minimizes the objective value of IQP (5.4), thus,

$$\Lambda(y^i_{vs})\le\Lambda({y^i_{vs}}')\le\sum_{v\in\mathscr{V}_i}\sum_{s\in[S]}(P^i_{vs}+{F^i_{vs}}^s){y^i_{vs}}'+V_i^2$$
$$\le\Lambda^*+\varepsilon V_i^2+V_i^2=\Lambda^*+(1+\varepsilon)V_i^2.$$ □

Theorem 12. A_{sub1} is an α-approximation algorithm where $\alpha = 1+(1+\varepsilon)cV_i$ and $c = \max_{v_1,v_2,s_1,s_2} \{P^i_{v_1,s_1}/P^i_{v_2,s_2}\}$.

Proof: The optimal objective value of IQP (5.4) is $\Lambda^* \geq \sum_{v\in\mathscr{V}_i}\sum_{s\in[S]} P^i_{vs}{y^i_{vs}}^* \geq \frac{V_i}{c}$ since $P^i_{vs} \geq \frac{1}{c}, \forall v,s$. According to Lemma 25, we have

$$\frac{\Lambda(y^i_{vs})}{\Lambda^*} \leq 1 + \frac{(1+\varepsilon)V_i^2}{\Lambda^*} \leq 1+(1+\varepsilon)cV_i. \qquad \square$$

5.4.3 A Heuristic Algorithm

Algorithm 11 A Greedy Algorithm A_{sub2}

Input: **Input**: $\Phi_i, \{P^i_{vs}\}, \{P^i_{v_1,v_2,s_1,s_2}\}, \{a^i_{vk}\}$

for all $v \in \mathscr{V}_i$ **do**
  $\mathbb{S} = [S]$;
  for all $s \in [S]$ **do**
    if $z^i_{ks} + a^i_{vk} > B_{ks}, \forall k \in [K]$ **then**
      $\mathbb{S} = \mathbb{S} \backslash s$;
    end if
  end for
  for all $s \in \mathbb{S}$ **do**
    Compute the increase $\Delta(y^i_{vs})$ of the objective function value of IQP (5.4) due the the assignment of $y^i_{vs} = 1$;
  end for
  $s^* = \arg\min_{s\in\mathbb{S}}\{\Delta(y^i_{vs})\}$;
  $y^i_{vs^*} = 1; z_{ks^*} = z^i_{ks^*} + a^i_{vk}, \forall k \in [K]$;
end for
$cost_i = \sum_{v\in\mathscr{V}_i}\sum_{s\in[S]} P^i_{vs}y^i_{vs} + \sum_{v_1,v_2\in\mathscr{V}_i}\sum_{(s_1,s_2)\in\mathbb{E}} P^i_{v_1,v_2,s_1,s_2}y^i_{v_1,s_1}y^i_{v_2,s_2}$; Save $\{y^i_{vs}\}$ to l^*;
Return $cost_i, l^*$.

We next introduce a heuristic algorithm that starts with an empty solution, and iteratively selects a container for greedy assignment to an available zone. At the v-th iteration, we simply choose the v-th container from set $\mathscr{V}_i$. Let $\Delta(y^i_{vs})$ denote the increment of the objective value of IQP (5.4) due to the assignment of $y^i_{vs} = 1$. To select a zone for container v, we first compute a candidate set $\mathbb{S}$ that includes all available zones. If the amount of allocated type-k resource in zone s has exceeded the bound B_{ks}, s is removed from $\mathbb{S}$. We continue to compute the value of $\Delta(y^i_{vs})$, for $s \in \mathbb{S}$, and choose s^* so that $\Delta(y^i_{vs^*}) = \min_{s\in Set}\Delta(y^i_{vs})$. We assign container v to zone s^*. The time complexity of this

algorithm is $O(VS)$ since we execute V_i iterations and evaluate S zones during each iteration. We summarize this algorithm in A_{sub2}. While we do not prove a performance guarantee, trace-driven simulations in Ch. 5.6 show that A_{sub2} can produce a 2-approximate solution in all cases tested.

5.5 Online Algorithm Design

Leveraging the cost minimization algorithms from Ch. 5.4 as a building block, we next present our online algorithm in Ch. 5.5.1. Upon the arrival of each request, the algorithm determines immediately whether to accept it, and if so, how to place its CC. Theoretical analysis is conducted in Ch. 5.5.2.

5.5.1 Online Algorithm Framework

Our main idea for the online algorithm design is as follows. We resort to the help of the classic primal-dual technique, and apply it to the compact-exponential ILP (5.2) and its dual (5.3). If request i's placement scheme l is accepted, then let $x_{il} = 1$, and update the corresponding variables x_i and y^i_{vs} in IP (5.1) according to l. Upon arrival of a request i, a set of primal variables $x_{il}, \forall l \in \zeta_i$ and the associated dual constraints ($b_{il} - \sum_{m\in\mathscr{M}} \sum_{t\in[t_i^-,t_i^+]} f^{il}_{m,t} p_{m,t}, \forall l \in \zeta_i$) appear. Complementary slackness requires that x_{il} remains zero unless constraint (5.3a) is tight for scheme l. Next, let's examine the right hand side (RHS) of constraint (5.3a). If we interpret dual variable $p_{m,t}$ as the price per unit of type-m resource at time t, then $\sum_{m\in\mathscr{M}} \sum_{t\in[t_i^-,t_i^+]} f^{il}_{m,t} p_{m,t}$ is the placement cost of request i's CC according to scheme l. The RHS of (5.3a) can be viewed as request i's utility with scheme l. If we interpret dual variable u_i as request i's utility, u_i can be assigned to the maximum of 0 and the RHS of (5.3a), *i.e.,*

$$u_i = \max\big\{0, \max_{l\in\zeta_i}\{b_{il} - \sum_{m\in\mathscr{M}} \sum_{t\in[t_i^-,t_i^+]} f^{il}_{m,t} p_{m,t}\}\big\}. \tag{5.7}$$

Accordingly, if $u_i > 0$, we accept request i; otherwise, we reject request i. The challenge lies in finding scheme l that maximizes the RHS of (5.3a), which is equivalent to finding the scheme that

minimizes the placement cost of request i's CC. Given $p_{m,t}$, it is easy to compute the computation cost P^i_{vs} and communication cost $P^i_{v_1,v_2,s_1,s_2}$. Then the problem becomes the one-shot CC placement problem we studied in Ch. 5.4, and can be formulated into an IQP (5.4). Assume that A_{sub} is an α-approximation algorithm for IQP (5.4). It returns a scheme l^* with cost $cost_i$. Then, we have $\frac{cost_i}{\alpha} \leq \sum_{m\in\mathscr{M}} \sum_{t\in[t_i^-,t_i^+]} f^{il}_{m,t} p_{m,t}, \forall l \in \zeta_i$. Let $u_i = \max\left\{0, b_i - \frac{cost_i}{\alpha}\right\}$, satisfying constraints (5.3a) for any $l \in \zeta_i$. If $u_i > 0$, request i is accepted and its CC is placed according to scheme l^*; if $u_i \leq 0$, request i is rejected.

We next discuss the update of dual variable $p_{m,t}$. Recall that $p_{m,t}$ represents the unit price of type-m resource at t. We define a new variable $z_{m,t}$ as the amount of type-m resource consumed at t, and let $p_{m,t}$ be a function of $z_{m,t}$, as follows:

$$p_{m,t}(z_{m,t}) = \lambda^{\frac{z_{m,t}}{C_m}} - 1, \forall m \in \mathscr{M}, \forall t \in [T], \tag{5.8}$$

where $\lambda = 2(\alpha U + 1)$. $p_{m,t}$ starts at zero and increases exponentially with the increase of resource consumption. $p_{m,t}$ is close to zero when resources are abundant, allowing CCs to consume resources freely. It grows quickly to a carefully designed large value λ when $z_{m,t}$ is close to the capacity C_m, so that the service provider will barely allocate any type-m resource to a CC, unless its valuation is sufficiently high.

A_{online^*} in Algorithm 12 is our online algorithm, which calls A_{sub} for each CC request to determine its placement scheme. Note that A_{online^*} can call either of the algorithms designed in the previous section (A_{sub1} and A_{sub2}), or any alternative that solves IQP (5.4), as its sub-routine. By default, all variables are set to zero. Line 1 initializes the value of λ, to prepare for the update of the dual variable $p_{m,t}$. Upon the arrival of a request i, we first compute the computation and communication costs when assigning containers to different zones in line 3. A_{sub} is executed for each request to compute a placement scheme l^* and the corresponding cost $cost_i$. If request i can obtain a positive utility in some scheme, it is accepted, and the corresponding primal and dual variables are updated (lines 6-8). We then increase the usage of resources and raise resource prices accordingly (lines 9-12).

Algorithm 12 A Primal-dual Online Framework A_{online^*}

Input: $\{\Phi_i\}, \{C_m\}, \{a^i_{vk}\}, U, \alpha$

1: Initialize $\lambda = 2(\alpha U + 1)$;
2: **Upon the arrival of the ith CC request**
3: Compute $\{P^i_{vs}\}$ and $\{P^i_{v_1,v_2,s_1,s_2}\}$ based on $\{p_{m,t}\}$;
4: $\left(cost_i, l^*\right) = A_{sub}\left(\Phi_i, \{P^i_{vs}\}, \{P^i_{v_1,v_2,s_1,s_2}\}, \{a^i_{vk}\}\right)$;
5: **if** $b_i - \frac{cost_i}{\alpha} > 0$ **then**
6: $\quad u_i = b_i - \frac{cost_i}{\alpha}; x_i = 1; x_{il^*} = 1$;
7: $\quad$ Update y^i_{vs} and $f^{il^*}_{m,t}$ according to option l^*.
8: $\quad$ Accept request i and allocate its CC according to y^i_{vs};
9: $\quad$ **for** $t \in [t_i^-, t_i^+]$ **do**
10: $\quad\quad z_{m,t} = z_{m,t} + f^{il^*}_{m,t}, \forall m \in \mathscr{M}$;
11: $\quad\quad p_{m,t} = \lambda^{\frac{z_{m,t}}{C_m}} - 1, \forall m \in \mathscr{M}$;
12: $\quad$ **end for**
13: **else**
14: $\quad$ Reject request i.
15: **end if**

5.5.2 Theoretical Analysis

Next we analyze properties of A_{online^*}, based on the assumption that A_{sub} can compute an α-approximate solution to IQP (5.4) in polynomial time.

i) Feasibility and Running time.

Lemma 26. A_{online^*} computes a feasible solution to IP (5.1) and one for ILP (5.2), respectively.

Proof: We first examine ILP (5.2). Constraint (5.2b) is satisfied because line 6 in A_{online^*} guarantees that only one option can be accepted. Next, we prove that the capacity constraint (5.2a) is never violated. Otherwise, let request i be the first accepted request that violates the capacity constraint of type-m resource at time t with option l^*. The amount of allocated type-m resource before request i arrives is: $z_{m,t} > C_m - f^{il^*}_{m,t}$. Under the assumption that $\frac{f^{il^*}_{m,t}}{C_m} \leq \frac{1}{\log \lambda}$, the price of type-$m$ resource at t for request i is:

$$p_{m,t} \geq \lambda^{1-\frac{f^{il^*}_{m,t}}{C_m}} - 1 \geq \lambda^{1-\frac{1}{\log \lambda}} - 1 \geq \frac{\lambda}{2} - 1 = \alpha U.$$

Thus, by $U \geq \frac{b_i}{f^{il^*}_m(t)}$, we can obtain

$$cost_i \geq p_{m,t} f^{il^*}_{m,t} \geq \alpha U f^{il^*}_m \geq \alpha b_i.$$

We further have $b_i - \frac{cost_i}{\alpha} \leq 0$, which contradicts the assumption that request i is accepted with $b_i - \frac{cost_i}{\alpha} > 0$. Thus, constraint (5.2a) holds.

We next investigate IP (5.1). Constraints (5.1c), (5.1d) and (5.1f) are satisfied by algorithm A_{sub}. In addition, the correspondence relation between IP (5.1) and ILP (5.2) guarantees constraints (5.1a), (5.1b), and (5.1e) hold. □

Lemma 27. A_{online^*} outputs a feasible solution to LP (5.3).

Proof: Let OPT_i be the optimal objective value of the subproblem in (5.4) for request i. OPT_i equals $\min_{l \in \zeta_i}\{\sum_{m \in \mathscr{M}} \sum_{t \in [t_i^-, t_i^+]} f_{m,t}^{il} p_{m,t}\}$. Because A_{sub} is an α-approximation algorithm that generates an objective value $cost_i$, we have $\frac{cost_i}{\alpha} \leq OPT_i$. If $b_i - \frac{cost_i}{\alpha} > 0$, A_{online^*} updates dual variable u_i in line 6, we obtain

$$u_i = b_i - \frac{cost_i}{\alpha} \geq b_i - OPT_i \geq b_i - \sum_{m \in \mathscr{M}} \sum_{t \in [t_i^-, t_i^+]} f_{m,t}^{il} p_{m,t}, \forall l \in \zeta_i.$$

Otherwise, $u_i = 0 \geq b_i - OPT_i$. Therefore, constraint (5.3a) holds for each request i and the lemma follows. □

Theorem 13. A_{online^*} generates feasible solutions for IP (5.1), ILP (5.2) and LP (5.3) in polynomial time.

Proof: Line 1 takes one step to compute the value of λ. Upon the arrival of request i, line 3 takes $O((SV_i)^2)$ steps to initialize the cost vector. A_{sub} in line 4 runs in polynomial time to compute placement cost. Within the body of the `if` statement, lines 6-8 update primal variables in $O(V_i S + (t_i^+ - t_i^- + 1)|\mathscr{M}|)$ steps. The complexity of the `for` loop in lines 9-12 is $O((t_i^+ - t_i^- + 1)|\mathscr{M}|)$. Therefore, the running time of A_{online^*} is polynomial. Combining Lemma 26 and Lemma 27, we finish the proof. □

ii) Competitive Ratio.

We next analyze the competitive ratio of A_{online^*}. The *competitive ratio* is the upper-bound ratio of optimal objective of IP (5.1) to the objective value achieved by A_{online^*}. We first introduce

a primal-dual analysis framework in Lemma 28, which guides the proof of the competitive ratio. We next define a *Resource-Price Relationship* for A_{online^*} in Definition 14 and the differential version of it in Definition 15, respectively. We prove that if the Resource-Price Relationship holds for a given β, A_{online^*} satisfies the inequality in Lemma 28. We then present the value of β in Lemma 30 and prove that A_{online^*} is $\alpha\beta$-competitive in Theorem 14.

Let OPT_1 and OPT_2 be the optimal objective values of IP (5.1) and ILP (5.2), respectively. We have $OPT_1 = OPT_2$. Let P_i and D_i denote the objective value of primal ILP (5.2) and that of dual LP (5.3) returned by A_{online^*} after processing request i. Let P_0 and D_0 be the initial values. A_{online^*} guarantees $P_0 = D_0 = 0$. Let P_I and D_I be the final primal and dual objective values achieved by A_{online^*}.

Lemma 28. If there exist two constants $\alpha \geq 1$ and $\beta \geq 1$ such that $P_i - P_{i-1} \geq \frac{1}{\alpha\beta}(D_i - D_{i-1}), \forall i \in [I]$, then the competitive ratio of A_{online^*} is $\alpha\beta$.

Proof: Summing up the inequalities for each request i, we have

$$P_I = \sum_i (P_i - P_{i-1}) \geq \frac{1}{\alpha\beta} \sum_i (D_i - D_{i-1}) = \frac{1}{\alpha\beta} D_I.$$

According to weak duality [24], $D_I \geq OPT_2$, hence, $P_I \geq \frac{1}{\alpha\beta} OPT_2 = \frac{1}{\alpha\beta} OPT_1$. The competitive ratio of A_{online^*} is $\alpha\beta$. □

We next define a *Resource-Price Relationship* and prove that if it holds for a given β, then the primal and dual objective values achieved by A_{online^*} satisfy the inequality in Lemma 28. Let $p^i_{m,t}$ denote the price of type-m resource at time t after handling request i. $z^i_{m,t}$ represents the amount of consumed type-m resource at time t after processing request i.

Definition 14. The Resource-Price Relationship for A_{online^*} with $\beta \geq 1$ is:

$$p^{i-1}_{m,t}(z^i_{m,t} - z^{i-1}_{m,t}) \geq \frac{1}{\beta} C_m (p^i_{m,t} - p^{i-1}_{m,t}), \forall i \in [I], \forall m \in \mathscr{M}, \forall t \in [t_i^-, t_i^+].$$

Lemma 29. If the Resource-Price Relationship holds for a given $\beta \geq 1$, then A_{online^*} guarantees that $P_i - P_{i-1} \geq \frac{1}{\alpha\beta}(D_i - D_{i-1}), \forall i \in [I]$.

Proof: If request i is rejected, then $P_i - P_{i-1} = D_i - D_{i-1} = 0$. Otherwise, we assume that request i is accepted and placed according to option l. The increment of the primal objective value is: $P_i - P_{i-1} = b_i$. Note that A_{online^*} assigns u_i to $b_i - \frac{1}{\alpha}\sum_{m\in\mathscr{M}}\sum_{t\in[t_i^-,t_i^+]} f_{m,t}^{il} p_{m,t}$ when request i with option l is accepted. Therefore,

$$b_i = u_i + \frac{1}{\alpha}\sum_{m\in\mathscr{M}}\sum_{t\in[t_i^-,t_i^+]} p_{m,t}^{i-1}(z_{m,t}^i - z_{m,t}^{i-1}).$$

The increase of the dual objective value is:

$$D_i - D_{i-1} = u_i + \sum_{m\in\mathscr{M}}\sum_{t\in[t_i^-,t_i^+]} C_m(p_{m,t}^i - p_{m,t}^{i-1}).$$

By summing up the Resource-Price Relationship over all $m \in \mathscr{M}$ and $t \in [t_i^-, t_i^+]$, we can obtain:

$$P_i - P_{i-1} \geq u_i + \frac{1}{\alpha\beta}(D_i - D_{i-1} - u_i).$$

Since $u_i \geq 0$ and $\alpha\beta \geq 1$, we have $P_i - P_{i-1} \geq \frac{1}{\alpha\beta}(D_i - D_{i-1})$. □

In order to compute the value of β, we make the following mild assumption and define the differential version of the Resource-Price Relationship based on it.

Assumption 1. The job demand is much smaller than the resource's capacity, *i.e.,* $f_{m,t}^{il} \ll C_m$.

Under Assumption 1, $z_{m,t}^i - z_{m,t}^{i-1}$ can be expressed as $dz_{m,t}$. The derivative of the Resource-Price Relationship under the above assumption is:

Definition 15. The Differential Resource-Price Relationship for A_{online^*} with $\beta \geq 1$ is:

$$p_{m,t}dz_{m,t} \geq \frac{C_m}{\beta}dp_{m,t}, \forall m \in \mathscr{M}, \forall t \in [t_i^-, t_i^+].$$

Lemma 30. $\beta = \ln\lambda$ and the price function defined in (5.8) satisfy the Differential Resource-Price Relationship.

Proof: The derivative of the price function is: $dp_{m,t} = \lambda^{\frac{z_{m,t}}{C_m}}\frac{\ln\lambda}{C_m}dz_{m,t}$. The Differential Resource-Price Relationship is:

$$(\lambda^{\frac{z_{m,t}}{C_m}} - 1)dz_{m,t} \geq \frac{C_m}{\beta}\lambda^{\frac{z_{m,t}}{C_m}}\frac{1}{C_m}\ln\lambda dz_{m,t}$$
$$\Rightarrow \beta \geq \ln\lambda\frac{\lambda^{\frac{z_{m,t}}{C_m}}}{\lambda^{\frac{z_{m,t}}{C_m}} - 1} \geq \ln\lambda.$$

Therefore this lemma holds for $\beta = \ln \lambda$. □

Theorem 14. The online auction A_{online^*} in Alg. 12 is $\alpha\beta$-competitive, where $\beta = \ln \lambda$ and α is the approximate ratio of A_{sub}.

Proof: Under Assumption 1, $dz_{m,t} = z^i_{m,t} - z^{i-1}_{m,t}$ is much smaller than the capacity of type-m resource (C_m), we have $dp_{m,t} = p'_m dz_{m,t} = p^i_{m,t} - p^{i-1}_{m,t}$. As a result, we can conclude that the Resource-Price Relationship holds for $\beta = \ln \lambda$. Then, combining Lemma 28 and Lemma 29, we finish the proof. □

5.6 Performance Evaluation

We evaluate the performance of our one-shot algorithms A_{sub1}, A_{sub2} and online algorithm A_{online^*} through trace-driven simulation studies. We first introduce the simulation setup for evaluation of the two one-shot algorithms. The default number of zones is set to 9 according to the number of Google data centers in the United States [67]. We exploit Google Cluster Data version 1 [3], and configure each CC according to each job's information in the trace. We assume that each CC contains 2-8 containers and consumes two types of computational resource, since the trace data only includes resource demands for CPU and RAM. The resource consumption a^i_{vk} is set according to the resource demand of each subtask in the trace [3]. The traffic volume between containers $\Delta^i_{v_1,v_2}$ is randomly generated within a range of $[0,10]$. The cost P^i_{vs} and $P^i_{v_1,v_2,s_1,s_2}$ are randomly drawn from $[0,1]$. The default value of B_{ks} is 10. For the online setup, we assume each time slot is 5 minutes and the system spans 100 slots by default. Each request's start and end times are set based on each job's timestamp in the trace. The resource capacities, C_{ks} and D_{s_1,s_2} are randomly generated within $[50,100]$. The request value b_i is randomly chosen from an interval determined by U, whose default value is 50. We repeat each set of simulations 20 times, and use the average result to plot the corresponding figure.

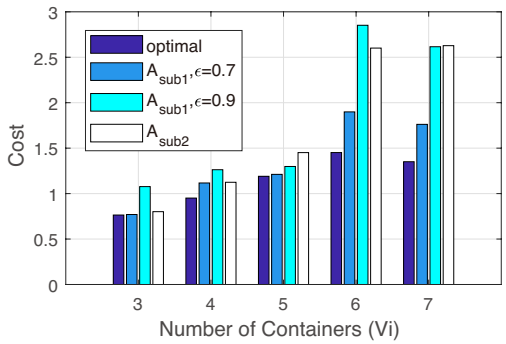

Figure 5.2: Cost of A_{sub1} and A_{sub2} under different values of V_i.

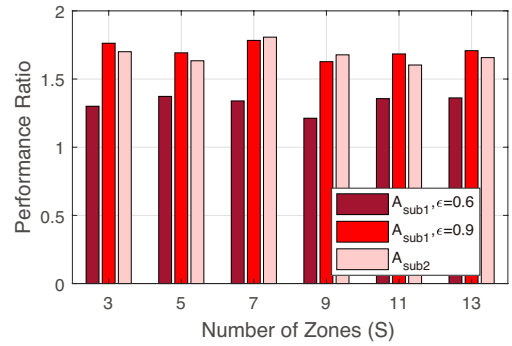

Figure 5.3: Performance ratio of A_{sub1} and A_{sub2} under different values of S.

5.6.1 Performance of A_{sub1} and A_{sub2}

Cost and Performance Ratio. Fig. 5.2 compares the total cost produced by A_{sub1} and A_{sub2} with the optimal cost under different numbers of containers. We can observe that the gap between the cost of A_{sub1} and the optimal cost becomes larger when the number of containers increases, and gets smaller when the value of ε decreases, which is in line with the analysis in Lemma 25. In addition, A_{sub1} achieves a lower cost than A_{sub2} when we input a smaller ε.

We next examine the *performance ratio*, measured by the ratio of the objective value of IQP (5.4) generated by our algorithms to the optimal objective value of IQP (5.4). We fix the number of containers to 5, and plot the performance ratios of A_{sub1} and A_{sub2} in Fig. 5.3. It can be observed that both A_{sub1} and A_{sub2} perform well with a low performance ratio (< 2). The value of S has little impact on the performance of A_{sub1} while the value of ε is related to the ratio, echoing Theorem 12. A_{sub1} outperforms A_{sub2} when ε is relatively small (0.6). We further modify the number of containers and plot the ratios in Fig. 5.4. The ratio increases with the growth of the number of containers, validating the analysis in Theorem 12 that the value of V_i determines the approximate ratio. Moreover, when there is more than one type of computational resource, A_{sub2} also works well and the ratio is smaller than 2.

Time Complexity. We apply the tic and toc functions in MATLAB to measure the execution time of the main program of A_{sub1} and A_{sub2} without counting the initialization stage. We run 20 tests

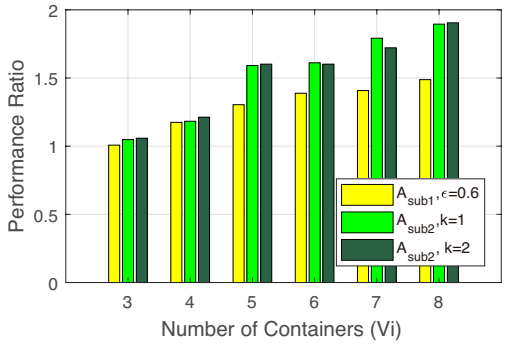

Figure 5.4: Performance ratio of A_{sub1} and A_{sub2} under different values of V_i and K.

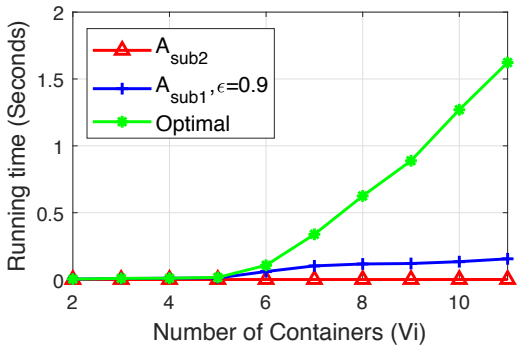

Figure 5.5: The average running time of A_{sub1} and A_{sub2} with different V_i.

on a laptop computer (Intel Core i7- 6700HQ/16GB RAM) and present the average result in Fig. 5.5. We implement the optimal one-shot algorithm by listing all possible placement schemes. We can observe that both A_{sub1} and A_{sub2} run much faster than the optimal algorithm. The running time grows with an increasing V_i, and the observed values are below 0.5 seconds.

5.6.2 Performance of A_{online^*}

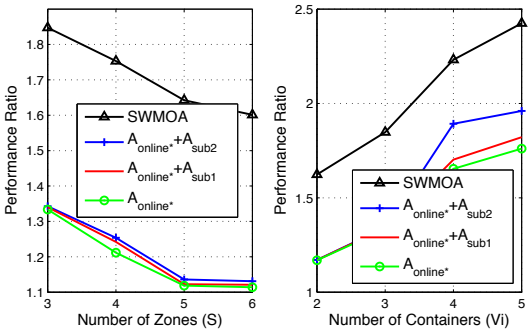

Figure 5.6: Performance ratio of A_{online^*} and SWMOA in [61] under different S and V.

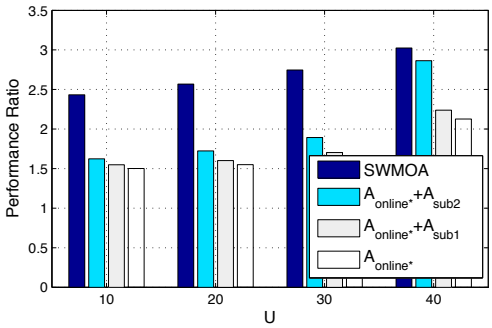

Figure 5.7: Performance ratio of A_{online^*} and SWMOA in [61] with different U.

Performance Ratio and Objective Value. We first examine the performance ratio of A_{online^*}, when we plug in the sub-algorithm that exactly solves IQP (5.4) (labeled by A_{online^*}), A_{sub1} (labeled by $A_{online^*} + A_{sub1}$) and A_{sub2} (labeled by $A_{online^*} + A_{sub2}$). The *performance ratio* of A_{online} is the ratio of the optimal objective value of IP (5.1) to the objective value of IP (5.1) generated by A_{online^*}. Based on the observation from the above subsection, we set $\alpha = 2$ for both A_{sub1} and

A_{sub2}. We fix the number of containers in each CC to 3 and the number of CC requests to 100, but vary the number of zones. The results are plotted in the left of Fig. 5.6. We observe that the ratio drops sharply with the increase of S, but remains steady when $S \geq 5$. This is because more zones bring more placement options. As a result, each CC request has a high probability of being accepted by A_{online^*}, leading to a better performance. When S is large enough, the ratio is dominated by other parameters, *e.g.,* V_i and U. The right of Fig. 5.6 illustrates that a lower ratio comes with a smaller number of CCs. Comparing Fig. 5.3 and Fig. 5.4 with Fig. 5.6, we can conclude that our online algorithm framework incurs only a small loss in performance ratio. In Fig. 5.7, we examine the impact of two parameters: U and I. Again, A_{online^*} with the optimal sub-algorithm has the best performance. The observed ratios are better than the theoretical worst-case bound and remain at a low level. The left figure shows that the performance ratio is larger for a larger value of U. The theoretical competitive ratio proven in Theorem 14 implies this result. The ratio fluctuates with the number of requests in the right figure, which indicates that the value of I does not have a major influence on the ratio.

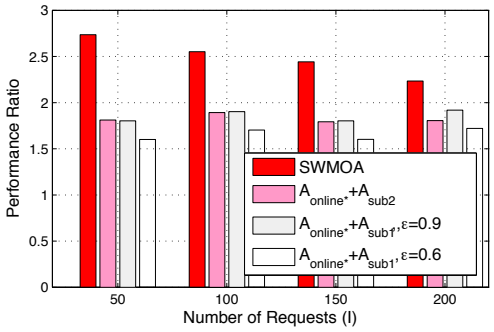

Figure 5.8: Performance ratio of A_{online^*} and SWMOA in [61] with different I.

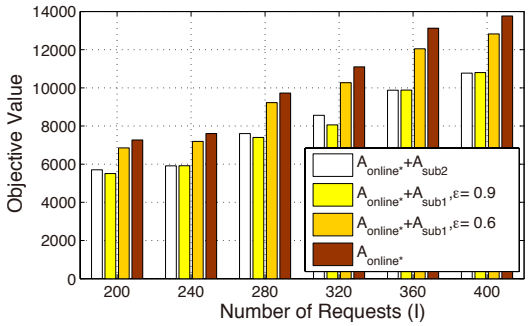

Figure 5.9: Objective Value achieved by A_{online^*}.

We further compare our online algorithm with Shi *et al.*'s online algorithm [61], SWMOA, which also makes decisions based on the current resource prices. Their price function depends on the number of resources and the number of time slots. We have implemented their algorithm and evaluated it using the same trace data. In Fig. 5.6, we can observe that our online algorithm can achieve a much lower performance ratio than SWMO, under different values of S and V_i. In

Fig. 5.7 and Fig. 5.8, we vary the value of another two parameters, U and I, and still obtain the same observation that our online algorithm always outperforms SWMOA.

We next investigate the objective value of ILP (5.2) achieved by A_{online^*}. Fig. 5.9 reflects that there is an upward trend with a larger number of requests. The underlying reason is that A_{online^*} can select more high-value requests from a large set of participants. Similar to the observation in Fig. 5.7, the objective value achieved by $A_{online^*} + A_{sub1}$ with a small ε is higher than that of $A_{online^*} + A_{sub2}$.

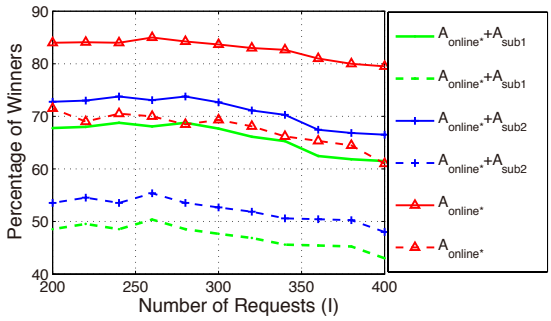

Figure 5.10: Percentage of winners under different I.

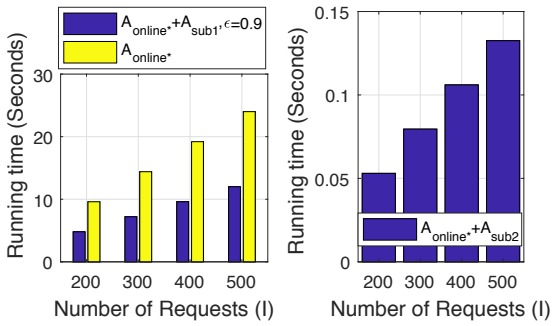

Figure 5.11: The average running time of A_{online^*}.

Winner Satisfaction and Time Complexity. *User satisfaction*, which is measured by the percentage of winners, is shown in Fig. 5.10. The three solid lines represent the percentages of winners when $C_{ks} = 100$, and the three dot lines are for the case of $C_{ks} = 50$. We can see that the percentage of winners drops when a high number of CCs wait for deployment. The reason can be explained as follows: The number of winners remains relatively steady when the resource capacity is fixed. Therefore, only a small percentage of CC requests can be selected from a large set. The resource capacity influences the number of winners. Thus, a higher percentage of winners comes with a large capacity. We next fix the number of containers in each CC to 5. Fig. 5.11 shows the average running time of our online algorithms with varying number of requests. Again, the shortest running time is observed when we call A_{sub2} in our online algorithm. A_{online^*} with A_{sub1} has a slightly longer running time, followed by A_{online^*} with the optimal one-shot algorithm.

5.7 Summary

In this Chapter, we proposed an efficient online algorithm for placing container clusters in cloud zones, considering both container deployment and the demand of inter-container traffic. We first leverage exhaustive sampling and ST rounding techniques to design a one-shot algorithm that determines the placement scheme for the current CC. Then we apply compact-exponential and the online primal-dual techniques to design an online algorithm framework that uses the one-shot algorithm to make on-spot decisions based on resource prices. Our online algorithm achieves computational and economical efficiencies.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Cloud computing has attracted more and more attention in the community of computer science and information technology, for delivering on-demand computing resources over the Internet. Recent cloud platforms allow users to request tailor-made VMs or containers that bundle various resources at user-specified amounts [7] [2]. How to dynamically allocate cloud resources becomes a fundamental and important problem. Successes in designing dynamic resource provisioning in clouds will help current cloud service providers to utilize their computing, storage and communication resources more efficiently, to improve system reliability, and to obtain a higher social welfare. In this thesis, we propose efficient algorithms for cloud resource allocation and workload dispatching, contributing towards the efficient and robust operation of cloud systems.

- In Chapter 3, we study the auction design for dynamic VM provisioning. We focus on cloud jobs with soft deadlines that arrive in an online fashion. We adapt the classic primal-dual framework for efficient approximation algorithm design, and employ a posted-pricing framework for truthful online mechanism design, to derive truthful online auctions that run efficiently and approach optimal social welfare. However, it turns out that the above techniques alone are not sufficient. A salient feature of Chapter 3 is the new compact-exponential optimization technique we introduce, which works in concert with a dual oracle to handle job completion time constraints imposed by hard and soft deadlines. Our compact-exponential method may further shed light on other algorithm and mechanism design scenarios where the optimization problem contains both conventional and non-conventional con-

straints, such as delay-constrained optimization in cyber physical systems.

- In Chapter 4, we present scheduling frameworks for cloud container provisioning under both offline and online settings. Our problem model is expressive enough to accommodate complex cloud computing jobs. We first apply the compact-exponential technique to reformulate the problem. Our offline algorithm solves the compact-exponential ILP fractionally, and adopt randomized rounding to obtain an integer solution. Our online algorithm implements the online learning technique to gradually solve the expected version of the compact-exponential ILP and its dual. Our offline and online algorithms achieve computational and economical efficiencies. The compact-exponential technique enjoyed another application in this chapter, suggesting its potential in a range of optimization problems.

- In Chapter 5, we investigate the online placement algorithm design for cloud container cluster provisioning. We take both the deployment of containers and the demand of inter-container traffic into consideration. Our online placement scheme consists of a one-shot algorithm that determines the placement scheme for the current CC and an online algorithm framework that decomposes the online decision making into on-spot decisions based on resource prices. We leverage exhaustive sampling and ST rounding techniques to compute quality solutions to the one-shot problem, and further exploit compact-exponential to handle placement constraints and the online primal-dual techniques for guaranteeing a good competitive ratio. Our online algorithm achieves computational and economical efficiencies.

6.2 Future work

Through the studies of this thesis, we also identify some interesting problems and new directions for future study. We next detail them as follows:

- *An Online Placement Framework for Service Chains in Geo-Distributed Clouds.* In this thesis, we focus on the efficient algorithm design for dynamic cloud resource provisioning. An interesting direction would be to extend our investigation to a new version of the cloud system – the Network Function Virtualization (NFV) system. NFV is emerging as a new paradigm for providing virtualized network services through service chains of virtual network functions (VNFs) [39]. VNFs typically execute on virtual machines in a cloud infrastructure, which consists of geo-distributed cloud data centers. Compared to traditional cloud services, key challenges in virtual network service provisioning lie in the optimal placement of VNF instances while considering inter-VNF traffic in a service chain. Furthermore, the end-to-end delay of each flow to pass through a service chain should be minimized when a service chain is processed upon its arrival. It is natural to target an online placement framework to address the above challenges, with a goal of minimizing the deployment cost. The combination of the compact-exponential method and classic primal-dual techniques may prove effective again here.

- *An Offline Scheduling Algorithm for General Computing Jobs.* In Chapter 4, we proposed an offline scheduling algorithm for jobs with chain structures, and left jobs with general directed acyclic dependence graphs for future work. A natural direction for future research is to study the offline scheduling algorithm design for this type of jobs. The challenge lies in the design of a separation oracle for polytope P_i. A polynomial-time separation oracle to solve the utility maximization problem for general jobs with arbitrary dependence graph topology is required if we adopt the same offline technique. Since the computational complexity of the current offline algorithm is high, another direction is to design a new offline algorithm with a fast running time.

- *Auction Mechanisms for Cloud Container and Container Cluster Services.* In Chapters 4 and 5, we focused on the algorithm design for cloud container and container cluster provisioning. A pricing mechanism to charge the usage is missing. An auction-based mechanism enables agility and efficiency in the cloud market by transferring the pricing functionality to the invisible hand of the market, having resources sold to users who value them the most. For example, spot instance pricing in Amazon EC2 is among the first real-world implementations of auctions in cloud computing, which, however, doesn't guarantee truthful bidding. As future work, we aim to design cloud container auctions that elicit truthful bids, execute in very short time, and approach a close-to-optimal social welfare or provider's profit.

- *An Online Auction for C-RAN Resource Provisioning with Operation Cost.* During my PhD study, I also paid attention to 5G networks. Two infrastructure revolutions are being implemented in 5G. The first is virtualization based, centralized cloud processing. Base-band signals are sampled and transmitted through front-haul links to a mobile cloud, for processing by mobile base station instances deployed in an on-demand fashion. User and channel information are aggregated to the cloud, facilitating optimized decision making. The second is the separation of infrastructure ownership from service provisioning in cellular networks. The "Tower" company now specializes in deployment and maintenance of the cloud radio access network (C-RAN) infrastructure [57]. Mobile operators focus instead on their sole business of wireless service provisioning. Mobile operators lease C-RAN resources that include spectrum resources at remote radio heads, front-haul bandwidth and mobile base station instances. Without considering the operation cost of the tower company, we have proposed a truthful auction to solve the one-round C-RAN resource allocation problem [85]. One possible direction of my future work is to investigate online C-RAN auction design with the consideration of the operation cost. It is in-

teresting to generalize the compact-exponential technique and learning theories to solve online problems in mobile networks.

Bibliography

[1] *Amazon EC2 instance type*. http://aws.amazon.com/ec2/instance-types/.

[2] *CloudSigma*. https://www.cloudsigma.com/.

[3] *Google Cluster Data, TraceVersion1*. https://code.google.com/p/googleclusterdata/wiki/TraceVersion1.

[4] *Guide to Fiber Optics & Premise Cabling*. https://goo.gl/Yd8UsW.

[5] *Linode*. https://www.linode.com/.

[6] *Microsoft Azure*. http://azure.microsoft.com/en-us/?rnd=1.

[7] *ProfitBricks*. https://www.profitbricks.com/.

[8] S. Agrawal and N. R. Devanur. Fast algorithms for online stochastic convex programming. In *Proc. of ACM-SIAM SODA*, 2015.

[9] S. Agrawal, Z. Wang, and Y. Ye. A dynamic near-optimal algorithm for online linear programming. *Operations Research*, 62(4):876–890, 2014.

[10] Aliyun. *Container Service*. https://goo.gl/CnLfBQ.

[11] Amazon. *Amazon EC2 Container Service*. https://aws.amazon.com/ecs/.

[12] Amazon. *Amazon ECS Clusters*. https://goo.gl/3pbXwB.

[13] Amazon. *Regions and Availability Zones*. https://goo.gl/oVQknf.

[14] A. Ambari. *Recommended memory configurations for the mapreduce service*. https://ambari.apache.org/1.2.1/installing-hadoop-using-ambari/content/ambari-chap3-7-9a.html.

[15] A. Archer, C. Papadimitriou, K. Talwar, and É. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. *Internet Mathematics*, 1(2):129–150, 2004.

[16] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *Proc. of IEEE FOCS*, 1996.

[17] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proc. of ACM STOC*, 1995.

[18] Y. Azar, I. Kalp-Shaltiel, B. Lucier, I. Menache, J. S. Naor, and J. Yaniv. Truthful online scheduling with commitments. In *Proc. of ACM EC*, 2015.

[19] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. In *Proc. of IEEE RTSS*, 1991.

[20] A. Blum, A. Gupta, Y. Mansour, and A. Sharma. Welfare and profit maximization with production costs. In *Proc. of IEEE FOCS*, 2011.

[21] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[22] S. Brahma. *The ellipsoid algorithm for linear programming*. `https://goo.gl/ge0p6u`.

[23] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. In *Proc. of ESA*, 2005.

[24] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal: dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2-3):93–263, 2009.

[25] E. Chang and C. Yap. Competitive online scheduling with level of service. In *Proc. of International Computing and Combinatorics Conference*, 2001.

[26] F. YL Chin and S. PY Fung. Improved competitive algorithms for online scheduling with partial job values. In *Proc. of International Computing and Combinatorics Conference*, 2003.

[27] F. YL Chin and S. PY Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.

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

[29] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichỳ, and N. Vakhania. Preemptive scheduling in overloaded systems. In *Proc. of International Colloquium on Automata, Languages, and Programming*, 2002.

[30] X. Dai, Y. Wang, J. M. Wang, and B. Bensaou. Energy efficient virtual cluster embedding in public data centers. In *Proc. of IEEE GLOBECOM*, 2015.

[31] N. R. Devanur. Fisher markets and convex programs. *JACM*, 2010.

[32] G. Even and M. Medina. Online multi-commodity flow with high demands. In *Proc. of International Workshop on Approximation and Online Algorithms*, 2012.

[33] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proc. of ACM-SIAM SODA*, 2006.

[34] Forbes. *Cloud computing market projected to reach $411B by 2020*. `https://www.forbes.com/sites/louiscolumbus/2017/10/18/cloud-computing-market-projected-to-reach-411b-by-2020/#1b93927b78f2`.

[35] Google. *Container Clusters*. `https://goo.gl/7Jt8tC`.

[36] Google. *Container Engine*. `https://cloud.google.com/container-engine/`.

[37] A. Gopinathan. *Strategyproof auction design for network resource allocation*. PhD thesis, University of Calgary, 2011.

[38] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *Proc. of ACM SIGCOMM*, 2014.

[39] S. Gu, Z. Li, C. Wu, and C. Huang. An efficient auction mechanism for service chains in the nfv market. In *Proc. of IEEE INFOCOM*, 2016.

[40] A. Gupta and M. Molinaro. How the experts algorithm can help solve LPs online. *Mathematics of Operations Research*, 41(4):1404–1431, 2016.

[41] M. Hajjat, X. Sun, Y. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. *Proc. of ACM SIGCOMM*, 2011.

[42] Z. Huang and A. Kim. Welfare maximization with production costs: a primal dual approach. In *Proc. of the ACM-SIAM SODA*, 2015.

[43] P. Jaillet and X. Lu. Near-optimal online algorithms for dynamic resource allocation problems. *arXiv:1208.2596*, 2012.

[44] N. Jain, I. Menache, J. S. Naor, and J. Yaniv. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. *ACM Transactions on Parallel Computing*, 2(1):3, 2015.

[45] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *Proc. of ACM SoCC*, 2010.

[46] T. Kesselheim, A. Tönnis, K. Radke, and B. Vöcking. Primal beats dual on online packing LPs in the random-order model. In *Proc. of ACM STOC*, 2014.

[47] K. H. Kim, A. Beloglazov, and R. Buyya. Power-aware provisioning of virtual machines for real-time cloud services. *Concurrency and Computation: Practice and Experience*, 23(13):1491–1505, 2011.

[48] G. Koren and D. Shasha. Dˆover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24(2):318–339, 1995.

[49] X. Li, J. Wu, S. Tang, and S. Lu. Let's stay together: Towards traffic aware virtual machine placement in data centers. In *Proc. of IEEE INFOCOM*, 2014.

[50] Z. Liu, I. Liu, S. Low, and A. Wierman. Pricing data center demand response. In *Proc.of ACM SIGMETRICS*, 2014.

[51] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European journal of operational research*, 176(2):657–690, 2007.

[52] L. Lovász and M. D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[53] B. Lucier, I. Menache, J. S. Naor, and J. Yaniv. Efficient online scheduling for deadline-sensitive jobs. In *Proc. of ACM SPAA*, 2013.

[54] Microsoft. *Azure Container Service*. `https://azure.microsoft.com/en-us/services/container-service/`.

[55] Microsoft. *Azure Container Service Cluster*. `https://goo.gl/URvRNg`.

[56] Microsoft. *Batch feature overview for developers*. `https://goo.gl/bQql24`.

[57] China Mobile. C-RAN: the road towards green RAN. *White Paper, version 3.0*, 2013.

[58] P. Orlik and H. Terao. *Arrangements of hyperplanes*, volume 300. Springer Science & Business Media, 2013.

[59] P. Raghavan and R. Motwani. *Randomized algorithms*. Cambridge Univ. Press, 1995.

[60] RightScale. *Social gaming in the cloud: a technical white paper*, 2013.

[61] W. Shi, C. Wu, and Z. Li. An online mechanism for dynamic virtual cluster provisioning in geo-distributed clouds. In *Proc. of IEEE INFOCOM*, 2016.

[62] W. Shi, L. Zhang, C. Wu, Z. Li, and F. Lau. An online auction framework for dynamic resource provisioning in cloud computing. In *Proc. of ACM SIGMETRICS*, 2014.

[63] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1):461–474, 1993.

[64] Q. Wang, K. Ren, and X. Meng. When cloud meets ebay: towards effective pricing for cloud computing. In *Proc. of IEEE INFOCOM*, 2012.

[65] Wikipedia. *Convex conjugate*. `http://en.wikipedia.org/wiki/Convex_conjugate`.

[66] Wikipedia. *Dynamic frequency scaling*. `http://en.wikipedia.org/wiki/Dynamic_frequency_scaling`.

[67] Wikipedia. *Google data centers*. `https://goo.gl/oKYNri`.

[68] Wikipedia. *Karmarkar's algorithm*. `https://en.wikipedia.org/wiki/Karmarkar's_algorithm`.

[69] Wikipedia. *Poisson point process*. `https://en.wikipedia.org/wiki/Poisson_point_process`.

[70] Wikipedia. *Rendering pipeline overview*. `https://www.opengl.org/wiki/Rendering_Pipeline_Overview`.

[71] Wikipedia. *Topological sorting*. `https://en.wikipedia.org/wiki/Topological_sorting`.

[72] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[73] X. Xu, H. Yu, and X. Pei. A novel resource scheduling approach in container based clouds. In *Proc. of IEEE CSE*, 2014.

[74] R. Yu, G. Xue, X. Zhang, and D. Li. Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers. In *Proc. of IEEE INFOCOM*, 2017.

[75] S. Zaman and D. Grosu. Combinatorial auction-based dynamic VM provisioning and allocation in clouds. In *Proc. of IEEE Cloud CloudCom*, 2011.

[76] ZDNet. *Containers: fundamental to the cloud's evolution*. `https://goo.gl/PPWmxe`.

[77] H. Zhang, B. Li, H. Jiang, F. Liu, A.V. Vasilakos, and J. Liu. A framework for truthful online auctions in cloud computing with heterogeneous user demands. In *Proc. of IEEE INFOCOM*, 2013.

[78] L. Zhang, Z. Li, and C. Wu. Dynamic resource provisioning in cloud computing: a randomized auction approach. In *Proc. of IEEE INFOCOM*, 2014.

[79] L. Zhang, S. Ren, C. Wu, and Z. Li. A truthful incentive mechanism for emergency demand response in colocation data centers. In *Proc. of IEEE INFOCOM*, 2015.

[80] X. Zhang, Z. Huang, C. Wu, Z. Li, and F Lau. Online auctions in IaaS clouds: welfare and profit maximization with server costs. In *Proc. of ACM SIGMETRICS*, 2015.

[81] Y. Zheng, B. Ji, N. Shroff, and P. Sinha. Forget the deadline: scheduling interactive applications in data centers. In *Proc. of IEEE Cloud*, 2015.

[82] Z. Zheng and N. B. Shroff. Online multi-resource allocation for deadline sensitive jobs with partial values in the cloud. In *Proc. of IEEE INFOCOM*, 2016.

[83] R. Zhou, Z. Li, and C. Wu. Scheduling frameworks for cloud container services. *IEEE/ACM Transactions on Networking*, 26(1):436–450, 2018.

[84] R. Zhou, Z. Li, C. Wu, and Z. Huang. An efficient cloud market mechanism for computing jobs with soft deadlines. *IEEE/ACM Transactions on Networking*, 25(2):793–805, 2017.

[85] R. Zhou, X. Yin, Z. Li, and C. Wu. Virtualized resource sharing in cloud radio access networks: An auction approach. *Computer Communications*, 114:22–35, 2017.