Solving Short-term and Long-term Multi-resource Server Management Problems in the Cloud with Queueing and Combinatorial Models

by

Yuxiao Chen

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Solving Short-term and Long-term Multi-resource Server Management Problems in the Cloud with Queueing and Combinatorial Models

Yuxiao Chen
Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2022

Abstract

In recent years, an increasing number of companies and individuals use cloud services in their daily life. As their workload increases, the cloud providers pay more attention on managing their server capacities to reduce their expenses. In this thesis, we study the short-term capacity planning problem in data centers, and introduce a hybrid framework based on queuing model and combinatorial model. The framework calculates a deterministic server set with the minimum operational cost to satisfy the stochastic workload in a cloud. We embed the short-term framework into another framework that solves the long-term server capacity planning problem with longer planning horizon. The long-term framework calculates the periodic server purchasing plan to satisfy the workload in months or years with the minimum server purchasing cost and operational cost. Our experimental results indicate that both frameworks may calculate a solution to satisfy the workload in a reasonable runtime. Moreover, the solution is closer to the optimal when the workload is more stable.

Acknowledgments

First, I would like to thank my supervisors, Christopher Beck and Arik Senderovich, for their invaluable support on the thesis and my research life. They are great models for my research career. I appreciate the trust you have put in me.

I would also express my thankfulness to my committee member, Vahid Sarhangian, for the precious questions and comments on my thesis.

Thanks to all of my lab mates in TIDEL. To Alex, Eugene, Giovanni, Jason, Arnoosh, Louis, Ryo, Minori, Anton, Victor, Litong, Sonia, Lily, Oksana, Tan, and Luke, for welcoming me to this great lab and being a large family in campus. I enjoy the games, the meals, and the laughter we had together, and also thank to the conversations that help me to pursue my academic goal.

Finally, I would like to thank my family for their steady support and understanding.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Public and private clouds have become an important tool for more and more people. The clouds consist of data centers that provide all kinds of online services to customers (i.e. media, computation, storage, etc.). In recent years, the growing demand on cloud services has force the data centers to expand in both size and operational cost [8]. Due to the increasing operational cost, it is important to design a precise server management tool for data centers.

In this work, we focus on the server capacity planning problem in the clouds that provide infrastructure as a service (IaaS). The IaaS cloud assigns the multiple kinds of customer requests to virtual machines with the required resources, and the virtual machines are allocated to the different types of physical machines in the data center. The IaaS users are sensitive to the latency, so they sign a service level agreement (SLA) with the cloud provider to set a limitation on the cloud response time. The response time is measured as the time between the job arrival time and the start of its execution. An example of the SLA is that 99% of the virtual machines must have a response time less than 1 second. Thus, the capacity planning problem is to find the collection of servers that guarantees achievement of the SLAs while minimizing cost. The short-term problem assumes the workload stays the same, and the solver produces one set of servers that satisfies the workload with the minimum operating cost. In the long-term problem, the workload changes over time, so the solver has to make a periodic server purchasing plan to update the cloud capacity. The objective of the long-term problem becomes minimizing the total server purchasing cost and operating cost over the multi-period planning horizon.

The short-term capacity planning problem in cloud has been studied from multiple approaches, including queuing theory [31] and combinatorial optimization [70]. The cloud server management has been proved to be a challenging problem [15, 54], and two of the major challenges are evaluating the deterministic resource requirement

from a stochastic demands and finding the optimal scheduling strategy to satisfy the SLAs on the quality of service. In this work, we hybridize queueing theory based models and combinatorial optimization models to address both challenges.

The long-term capacity planning problem is commonly studied for the manufacturing and service industries [1]. Researchers have shown the computational difficulty of solving the planning problems with long planning horizon [5]. Thus, we approximate the optimal solution of the long-term problem by solving the planning problem in each period separately. Moreover, we prove that following the local optimal solution in each period achieves the long-term optimal in some special cases.

This thesis applies combinatorial optimization models including linear programming (LP) models and integer programming (IP) models to the short-term and long-term server capacity planning problems in the cloud. In the short-term problem, the IP model has to collaborate with queueing models to consider the stochastic behavior of the cloud. The experimental results show the combinatorial models can solve the planning problem in the real cloud (Google Cloud) in a reasonable time. However, the differences between the theoretical assumptions in the problems and the real life scenarios causes errors in the solutions. The cloud that fits better to the problem assumptions has a more accurate solution.

1.1 Thesis Outline

Chapter 2 presents the technical background of the mathematical tools we used in this thesis and reviews the related literature. The first section of the chapter explains the structure of general queueing models and presents the waiting time distribution of $M/M/n$ and $GI/GI/n$ queues. Then it formalizes the paradigm of the integer programming (IP) and its application in variable sized bin packing problem (VSBPP). The second section of the chapter reviews the studies about the short-term and long-term capacity planning problems. The works on the short-term problem are grouped based on their solution models, such as queueing models, combinatorial optimization models, and hybrid models. The long-term problem is rarely studied in cloud. Thus, the works on the management problems in other facilities with similar goals are reviewed, such as the hiring planning problem in service industries and the order planning problem in manufacturing industries.

Chapter 3 describes the hybrid framework for the short-term capacity planning problem. The framework solves the problem in two stages. In the first stage, it introduces service slots to represent the resources reserved for each class of jobs, and the number of service slots required for a given SLA is calculated by the queueing

models. The second stage uses an IP model to solve a variant of VSBPP that packs the service slots to a set of multi-type servers with the minimum operating cost. The IP model is relaxed to a linear programming (LP) model to reduce the computational complexity. The experimental results show that the linear relaxation significantly reduces the runtime and produces a high accuracy approximation of the IP optimal solution.

Chapter 4 is a case study of applying our short-term framework to the Google Cloud with Borg system. We use the trace of all jobs that arrived to one Google Cloud cluster in May 2nd, 2019 available from the public git repository [68] of Borg data. We observe the job data violates some assumptions in our framework. The framework assumes the jobs are classified by their resource requirements, but the Borg dataset does not include the class of the jobs, and none of the jobs has an exactly same resource requirement compared to another. Moreover, the problem assumes the jobs do not change their resource requirements while its execution, but the Borg system always allows the jobs to update their resource requirements. Thus, we group the jobs with similar resource requirements into the same class and round up their resource requirements to satisfy the assumptions in the framework, then apply the framework to approximate the optimal server requirement. The experimental results indicate that the framework can solve the capacity planning problem of Google Cloud in a reasonable time, and the quality of the approximation increases as we increase the number of job classes by classifying the jobs in more detail.

Chapter 5 extends the capacity planning problem to a long-term basis by adding the decision of purchasing servers as the workload changes through the multi-period planning horizon. The problem in cloud is similar to the problem in service industries, since both the servers and the employees are agents for the customers. Thus, we transform the dynamic programming (DP) model presented in Gans and Zhou's work on the employee management in call centers [21] to the cloud environment. Then the DP model is reformulated as a linear programming (LP) model. We decompose the LP model into multiple small LP submodels, where each submodel calculates the optimal server capacity in each period, and the local optimal solutions are combined to form a myopic long-term server purchasing policy. We prove this myopic policy is the optimal long-term server purchasing policy when the workload is non-decreasing. The empirical results support our theorem and show the computational complexity reduced by decomposing the LP model.

Chapter 6 concludes the results in this thesis and introduces some future directions for potential improvements.

1.2 Summary of Contributions

The main contributions of this thesis are listed below:

- We formally define the short-term and long-term server capacity planning problems in IaaS clouds based on the real cloud structure.
- We introduce a hybrid framework that uses queueing models and combinatorial optimization models to solve the short-term problem. It uses queueing models to ensure the performance of the cloud with a stochastic workload, and the combinatorial models solve the scheduling problem while considering the fragmentation loss caused by the imperfect resource allocation.
- We construct a dynamic programming (DP) model based on the existing model for the call center environment [21] to solve the long-term server capacity planning problem in cloud.
- We reformulate the DP model to a linear programming (LP) model, and then calculate a myopic solution by decomposing the LP model to approximate the optimal long-term solution.
- We prove the myopic solution achieves long-term optimal when the workload is non-decreasing through the planning horizon.

Chapter 2

Background

Our study combines queueing models and combinatorial models to solve server resource planning problems. The solution techniques introduced in this study are motivated and supported by past works [21, 32, 43, 58, 60, 67] in queueing theory and combinatorial optimization. Thus, we first present the theoretical background on the techniques that are used to solve our problems in Section 2.1. Then, Section 2.2 reviews recent studies on the cloud server management problem or problems with similar goals, including scheduling optimization in stochastic systems and resource management problems in service centers and in manufacturing industries.

2.1 Preliminaries

In this section, we present the fundamental theoretical background of queueing models and describe the waiting time of jobs in two types of queues: $M/M/n$ and $GI/GI/n$. Then we present the basis of integer programming (IP), a common combinatorial optimization technique that is used in our solutions. Finally, we explain the variable sized bin packing problem (VSBPP), since the problem we study in Section 3.3.3 is a variant of VSBPP.

2.1.1 Queueing Models

Queueing models [61] are mathematical representations that are constructed based on queueing theory to analyze the performance of queues. A queue, or a waiting line, is a system that has customers arrive, wait, receive services from agents, and leave stochastically (see Figure 2.1). The random variable A represents the inter-arrival time of two consecutive customers in a queue. The random variable L represents the time that a customer spends receiving services in a queue (also called lifetime). The waiting time of a customer in a queue is the time between when the customer

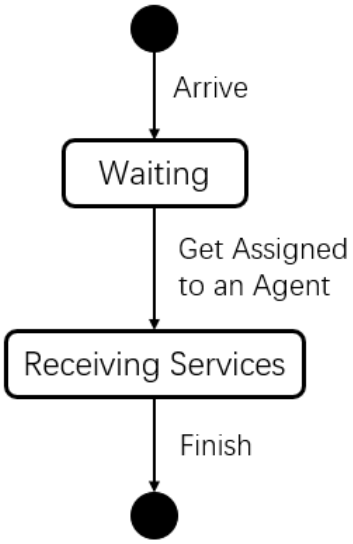


Figure 2.1: The life cycle of a customer in a queue system.

enters the queue and when the customer starts receiving services. If a customer is immediately assigned to an agent upon arrival, then the customer has zero waiting time. The waiting time of a customer is represented by the random variable W.

There are three basic parameters of a queue required by the queueing model: R_A states the distribution of the random variable that represents the customer inter-arrival times (A), R_L states the distribution of the random variable that represents the customer lifetimes (L), and n represents the number of agents that provide service in the queue system. The distribution types of the customer inter-arrival time (A) are denoted as follows: M stands for Markovian, and $R_A = M$ denotes that A follows an exponential distribution; D stands for deterministic, and $R_A = D$ denotes that A is deterministic; G stands for general, and $R_A = G$ denotes that there is no assumption on the distribution of A. The distribution types of the customer lifetimes (L) are denoted by R_L with the same notation.

A typical notation for the queues is Kendall's notation [33], which describes the structure of the queue in form of $R_A/R_L/n$. For example, in an $M/G/1$ queue, the inter-arrival times of the customers follow an exponential distribution, the lifetimes of the customers are arbitrarily distributed, and there is one agent in the system. In this work, we use two kinds of queueing models to analyze the waiting time distribution of cloud systems with different capacities: $M/M/n$ and $GI/GI/n$. In cloud, jobs play the role of the customers.

$M/M/n$ Queues

As noted, in $M/M/n$ queues the inter-arrival times of jobs follow an exponential distribution and the lifetimes follow another exponential distribution. The term n in the notation denotes that the queue may have more than one agent to provide services. In past works [64, 69], similar assumptions have been made for cloud systems.

In queueing theory, the performance of an $M/M/n$ queue is well-studied [58]. Suppose an $M/M/n$ queue has jobs arriving with expected inter-arrival time of $\frac{1}{\lambda}$ seconds and expected lifetime of $\frac{1}{\mu}$ seconds. According to the Chapter 2.7 in Sztrik [58], the probability of a job waiting for more than t seconds in this queue is,

$$PW\left(\frac{1}{\lambda}, \frac{1}{\mu}, t, n\right) = \gamma(\rho, n)e^{-(n-\rho)\mu t}, \tag{2.1}$$

where

$$\rho = \frac{\lambda}{\mu} \tag{2.2}$$

and

$$\gamma(\rho, n) = \frac{\frac{\rho^n}{n!} \cdot \frac{n}{n-\rho}}{\sum_{j=0}^{n-1} \frac{\rho^j}{j!} + \frac{\rho^n}{n!} \cdot \frac{n}{n-\rho}}. \tag{2.3}$$

Notice $\gamma(\rho, n)$ is the Erlang-C formula [19], which represents the probability of delay in the $M/M/n$ queue.

$GI/GI/n$ Queues

We generalize beyond the $M/M/n$ queue by removing the assumptions on the distributions of the random variables that represent the inter-arrival times and lifetimes of the jobs. Without these assumptions, the $M/M/n$ queue becomes a $GI/GI/n$ queue, which means the job arrivals and the lifetimes of the jobs are identically and independently distributed (i.i.d.), but follow any distribution. $GI/GI/n$ queues are too complex to have a closed formula representing the probability distribution of the waiting times of the jobs (see Chapter 4 in Gautam [22]). Fortunately, some approximations of the waiting time distributions in $GI/GI/n$ queues can be calculated with closed formulas [67].

Let $C(\cdot)$ be the coefficient of variation function, so for random variable X

$$C(X) = \frac{\sigma_X}{E(X)}$$

where $E(X)$ is the expected value of X and σ_X is the standard deviation of X.

Consider a $GI/GI/n$ queue that has jobs arriving with expected inter-arrival time $\frac{1}{\lambda}$ seconds and with an expected lifetime $\frac{1}{\mu}$ seconds. According to Whitt [67], with the assumption of heavy traffic, the probability distribution of a job waiting for more than t seconds in this queue can be approximated as follows,

$$PW\left(\frac{1}{\lambda}, \frac{1}{\mu}, t, n\right) \approx \frac{1}{\mu}\gamma(\rho, n) \exp\left(-(n-\rho)\mu t \cdot \frac{2}{C(A)^2 + C(L)^2}\right) \tag{2.4}$$

where A is the random variable representing the inter-arrival times, L is the random variable representing the lifetimes, and ρ and $\gamma(\rho, n)$ are defined in Equations 2.2 and 2.3. The detailed derivation of this formula can be found in Appendix A.

2.1.2 Integer Programming

In Section 3.3.3, we introduce an integer programming model to solve the optimization problem associated with packing jobs onto computational servers in the cloud. Integer programming (IP) is an exact technique for solving optimization problems with only discrete decision variables [32]. A generic IP model is formulated as follows:

$$\min\ c^T x \tag{2.5}$$

$$\text{s.t. } Ax \leq b \tag{2.6}$$

$$x \in \mathbb{Z}_+^n \tag{2.7}$$

In the model above, c is a vector of n objective function coefficients, x is a vector of n integer decision variables, and their dot product is the objective value (2.5) that the model is minimizing. Inequalities (2.6) are the constraints that each feasible solution must satisfy, where $A \in \mathbb{R}^{m \times n}$ is a matrix and b is a vector of m real numbers. An IP solver has to find the optimal x values that minimize the objective value across all feasible x.

Solving an IP model is NP-hard, so a common technique is to solve the linear relaxation first [43], since linear programming (LP) models can be solved in polynomial time [34]. The linear relaxation transforms an IP model into an LP model by expanding the domain of the decision variables (2.7) to $x \in \mathbb{R}_+^n$.

With the linear relaxation, a common algorithm for solving IP models is the branch-and-bound (B&B) algorithm [37]. The B&B algorithm explores the entire search space of the IP model by examining the nodes in a search tree. Given the generic IP model (2.5)-(2.7) as the root node N_{root} of the search tree, the B&B algorithm will find the optimal solution of the model in following steps:

1. Initialize an empty queue Q
2. $Q \leftarrow N_{root}$
3. $best\_sol = None$
4. $min\_cost = \infty$
5. while Q is not empty:
6. $\quad N = Q.pop()$
7. $\quad curr\_sol, curr\_cost = solveLP(N)$
8. $\quad$ if $curr\_cost < min\_cost$:
9. $\quad\quad$ if all decision variables in $curr\_sol$ are integers:
10. $\quad\quad\quad best\_sol = curr\_sol$
11. $\quad\quad\quad min\_cost = curr\_cost$
12. $\quad\quad$ else:
13. $\quad\quad\quad$ Pick a decision variable x_i that has a non-integer value x_i^* in $curr\_sol$
14. $\quad\quad\quad N_l, N_r = split\_by(N, x_i, x_i^*)$
15. $\quad\quad\quad Q \leftarrow N_l, N_r$
16. return $(best\_sol, min\_cost)$

In the pseudocode above, line 1 initializes the queue for open nodes, and line 2 inserts the root node into the queue as a starting point of the searching process. Line 3 and 4 initialize the variables for the best integer solution ($best\_sol$) that the algorithm has seen and its corresponding objective value (min_cost). Line 5-15 are the main search process of the B&B algorithm. While the queue of open nodes Q is not empty, the algorithm pops out the first node N in the queue in line 6. The function $solveLP(\cdot)$ in line 7 takes an IP model and solves the LP relaxation of it. The function returns the optimal solution and its corresponding objective value of the LP model as the variables $curr\_sol, curr\_cost$ respectively. If the linear relaxation is infeasible, then the function $solveLP(\cdot)$ returns $(None, \infty)$ to guarantee the condition in line 8 is false, and the node is discarded. If the linear relaxation solution has an objective value greater than or equal to min_cost, then there is no better solution in the search space defined by node N. Hence, the algorithm skips line 9-15 and moves to the next node in queue. When $curr\_cost < min\_cost$, if the $curr\_sol$ is an integer solution, then the new best integer solution is found and stored. If there exists a non-integer value x_i^* in $curr\_sol$, then we split the node N into two IP models as children of N. Suppose N is the IP model

$$\begin{aligned} \min\ & c^T x \\ \text{s.t. } & Ax \leq b \\ & x \in \mathbb{Z}_+^n \end{aligned}$$

and $x_i = x_i^*$ in the linear relaxation solution *curr_sol*. The function $split\_by(N, x_i, x_i^*)$ generates and returns two IP models:

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & Ax \leq b \\ & x_i \leq \lfloor x_i^* \rfloor \\ & x \in \mathbb{Z}_+^n \end{aligned}$$

and

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & Ax \leq b \\ & x_i \geq \lceil x_i^* \rceil \\ & x \in \mathbb{Z}_+^n. \end{aligned}$$

These two IP models are inserted into the queue Q as open nodes in line 15. Finally, after searching all open nodes, the B&B algorithm returns the best solution in the entire search space of N_{root} and its corresponding objective value $(best\_sol, min\_cost)$. If $best\_sol = None$, then the original IP model N_{root} is infeasible.

2.1.3 Variable Sized Bin Packing Problem

In Section 3.3.3, we solve a variant of the variable sized bin packing problem (VSBPP) [14, 20]. A VSBPP is defined as follows. Suppose there is a set of items $I = 1, ..., m$, where each item $i \in I$ has a weight of w_i and a set of bins $J = 1, ..., n$, where each bin $j \in J$ has a capacity of b_j. A bin can only contain a set of items $I_j \subseteq I$ with their total weight less or equal to b_j, i.e. $\sum_{i \in I_j} w_i \leq b_j$. We say that the items in I_j are packed into the server j. The VSBPP is solved by optimizing an objective value while packing every item into a bin.

There are a number of variations of VSBPP that have been studied. Friesen and Langston [20] addressed VSBPP with the objective of minimizing the total capacity associated with the bins. The authors proposed three approximation algorithms giving asymptotic worst case lower bounds. Correia, Gouveia, and Saldanha-Da-Gama [14] looked at the VSBPP with variable bin costs, so each bin $j \in J$ has an additional parameter f_j representing the cost of the bin. Their objective was to minimize the total cost of the bins that are not empty, and they propose two formulations of the optimization model for this problem.

The problem we define in Section 3.3.3 has an objective of minimizing the total

operating cost of the used servers (bins), while packing every service slot (item) into a server. This problem is similar to the VSBPP with variable bin costs, since different servers may have different operating costs. Thus, we present the integer programming model for the VSBPP with variable bin costs and an objective of minimizing the total cost formulated in Correia et al. [14]. In their formulation, f_j represents the cost of each bin $j \in J$, and the goal of the problem is to pack all the items in I into a subset of bins in J with the minimum cost.

Consider the binary variables x_{ij} indicating the item i is packed in the bin j ($i \in I, j \in J$) and binary variables y_j indicating whether the bin j is used. Then the VSBPP with variable costs is formulated as the following integer programming model:

$$\min \sum_{j \in J} f_j y_j \tag{2.8}$$

$$\text{s.t.} \sum_{j \in J} x_{ij} = 1, \qquad \forall i \in I \tag{2.9}$$

$$\sum_{i \in I} w_i x_{ij} \leq b_j y_j, \qquad \forall j \in J \tag{2.10}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in I, j \in J \tag{2.11}$$

$$y_j \in \{0, 1\}, \qquad \forall j \in J \tag{2.12}$$

The objective function (2.8) minimizes the cost of the bins used for packing all the items. Constraints (2.9) ensure that each item i is packed exactly once. Inequalities (2.10) state that for each bin j, the total weight packed in it cannot exceed the capacity of the bin. Constraints (2.11)-(2.12) are domain constraints.

2.2 Literature Review

In this section, we explore the literature on the server management problems in clouds. There are works solving such problems with different approaches, including queueing theory based models, linear programming models, and constraint programming models. We compare the problem definitions in the works with different approaches and discuss the limitations of each solution method. Then, we review the studies that apply linear programming models and mixed integer programming models to the inventory management problems in manufacturing industries and service industries. We discuss the relevance of the inventory management problems to the server management problems with purchasing, which is rarely studied in cloud environment.

2.2.1 Cloud Computing

Public cloud service has become a reality in last decade. The early stage of the cloud computing paradigm and the major challenges were studied in detail in 2011 [65]. Here we discuss the different types of cloud and their settings mentioned in the book [65] that are related to our work.

The cloud systems can be differentiated by the type of service they provide: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). In this work, we focus on the resource planning problems for the IaaS providers. The IaaS providers offer virtualized computation and storage resources to the customers, where the resources are located in the physical machines in data centers. Deciding on the set of physical machines with respect to some objective while satisfying the customer demands is called the resource planning problem, which significantly affects the profits for the IaaS providers.

The customers and the providers usually sign service level agreements (SLA) to guarantee the quality of services (QoS) provided. There are multiple types of SLAs with different restrictions. For example, some SLAs force a set of jobs to execute on the same or different machines, and some SLAs post minimum response times of the servers. In practice, the cloud providers use load balancing techniques and admission control mechanisms to satisfy the SLAs.

Load balancing algorithms assign customer requests onto a set of physical machines so that the load is equally distributed among multiple machines [53]. The load balancing algorithms are categorized into the class-agnostic algorithms and the class-aware algorithms. The class-agnostic load balancing algorithms are agnostic to the type of each incoming request and the customer who sent it. In contrast, the class-aware load

balancing algorithms make decisions based on the type of the customer making the request and/or the type of the service request.

Admission control algorithms decide the set of requests that should be admitted into the server when the server experiences heavy loads [9, 12]. There are two types of the admission control algorithms: the QoS-agnostic algorithms and QoS-aware algorithms. The QoS-agnostic admission control algorithms do not pick the requests based on their SLAs. In contrast, the QoS-aware admission control algorithms prioritize the requests with more restricted SLAs to guarantee their QoS.

In the rest of the section we explore the literature that studies resource planning problems in cloud or the problems with similar characteristics. The resource planning problems are categorized into two types based on their planning horizon: the short-term resource planning problem and the long-term resource planning problem. Short-term resource planning problems focus on finding a resource capacity to satisfy a given workload with respect to some objectives. In contrast, the long-term resource planning problems consider multiple workloads in different time periods and decides the amount of resources to purchase to maintain the quality of service across all time periods. Section 2.2.2 reviews the methods for the short-term resource planning problems, then Section 2.2.3 reviews the works on the long-term resource planning problems.

2.2.2 Short-term Resource Planning Problem

Short-term resource planning problems focus on finding the optimal server capacity and the scheduling strategy of the jobs in a short period. In recent years, increasing number of works had studied the resource management problem in cloud from different approaches, including combinatorial optimization and queueing theory [42].

The short-term resource planning problems can be classified into six categories based on their objectives: cost aware resource planning, energy aware resource planning, load balancing aware resource planning, quality of service (QoS) aware resource planning and utilization aware resource planning. In our work, we study a problem with an objective of minimizing the operating cost while satisfying a SLA that restricts the response time for the jobs. In the six categories, cost aware problems minimize the different sources of costs in the cloud, such as maintenance costs, delay penalty, and power costs. The energy aware problems minimize the energy consumption in the cloud, which is a major component in the operating cost. The QoS aware problems try to reach the highest level of QoS by maximizing the throughput or minimizing the response time for each job. We focus on the works about cost aware

Table 2.1: An overview of the problem assumptions in the cloud management literature with queueing models.

Literature	Problem Assumptions			
	Objective	Server	Job	SLA
[18]	energy consumption minimization	homogeneous servers	homogeneous jobs	response time
[28]	performance and energy consumption evaluation	homogeneous servers	homogeneous jobs	no SLA
[17]	profit maximization	homogeneous servers	heterogeneous jobs	tardiness upper bound on each job
[35]	performance evaluation	homogeneous servers	homogeneous jobs	no SLA
[11]	performance evaluation	homogeneous servers	homogeneous jobs	no SLA

resource planning, energy aware resource planning, and QoS aware resource planning, since they have a similar objective to our framework.

The literature has different assumptions on the servers and jobs in the cloud. The cloud may contain the same type of servers or multi-type servers. Similarly, the jobs arriving to the cloud are assumed to be identical or varied in different works. We assume the cloud has heterogeneous servers, and each server can process multiple heterogeneous jobs simultaneously as long as the resource capacity of the server is not violated. This is a common assumption for the clouds as cloud providers want to support more types of services, while using the servers from different generations.

The short-term resource planning problems in the cloud are widely studied with different mathematical models. We first review the works with queueing theory based algorithms. Queueing theory have been applied to analyze the performance and find the resource requirements with respect to the stochastic workload in many clouds. In contrast, the operating research community use combinatorial optimization models, such as linear programming (LP) models and constraint programming (CP) models, to optimize the job-to-machine assignment while considering the jobs with deterministic arrival and service time. After the discussion of combinatorial optimization works, we present the studies that use hybridized algorithms to take the advantages of multiple techniques.

Table 2.2: An overview of the solution techniques in the cloud management literature with queueing models.

Literature	Solution Techniques		
	Solution Model	Load Balancing	Admission Control
[18]	M/M/1/C queue for the load balancing algorithm and M/M/n/C queue for each server	class-agnostic	QoS-agnostic
[28]	M/M/n/C queue for each server	class-agnostic	QoS-agnostic
[17]	M/G/1 queue for each job class	no load balancing	QoS-agnostic
[35]	M/G/n queue for the cloud	class-agnostic	QoS-agnostic
[11]	M/G/n/C queue for the cloud	class-agnostic	QoS-agnostic

Queueing Theory Based Algorithms

A variety of types of queueing models have been used to represent the cloud systems and analyze their stochastic behaviors [31]. Tables 2.1 and 2.2 show the problem definitions and the solution models of the short-term resource planning literature with queueing models. The papers are ordered by their model complexity from the simplest to the most complicated, and the works with the same solution model are listed in chronological order. In this section, we review some works that use the queues with four parameters Kendall's notation $R_A/R_L/n/C$, where the fourth parameter C denotes the maximum capacity of the queue, and the queue is assumed to have infinite capacity if the fourth parameter is not shown (e.g. $M/M/n$).

Multiple studies have used $M/M/n/C$ queues to model the structure of cloud systems [18, 28]. With the $M/M/n/C$ queue assumption, the job arrivals are assumed to follow a Poisson process, the job service times follow an exponential distribution, the cloud system contains multiple servers to provide services, and a new job is rejected when there are C jobs waiting in the queue. Kafhali and Salah [18] model the cloud system in two levels. In the first level, an $M/M/1/C$ queue is used to represent the load balancing (LB) system. A new coming job is queued in the LB system if the waiting room for all servers are full, and the new job is rejected if the queue for the LB system is also full. The second level has an $M/M/n/C$ queue representing each physical machine, where each machine can process n jobs simultaneously and has a waiting room with finite size C. Using queueing theory, the authors first calculate the probability of a job being rejected by the LB system, that is, the probability of loss, then they calculate the mean queue length and the mean response time of

each server. In 2019, Hanini, Kafhali, and Salah [28] develop a data center manager (DCM) by modeling each server in the cloud as an $M/M/n/K$ queue. The DCM is designed to assign fewer jobs to the servers with queue length greater than $th < K$. The authors derive the performance parameters in the cloud with the DCM, such as the probability of loss, the mean response time, and the power consumption. The numerical results show the DCM helps the system to decrease these three measures.

Some works have fewer assumptions on the distribution of the job service time and use $M/G/\cdot/\cdot$ queues to analyze clouds [17, 35]. The $M/G/\cdot/\cdot$ queues only assume the job arrivals are following a Poisson process, and the service time may have any distribution with a known expected value and variance. Dutta et al. [17] classfy the jobs based on their priority, then model the queue for the jobs in each class as an $M/G/1$ queue. The queue for each job class is assumed to have one server that contains all the resources reserved for jobs in this class, and the server with more resources has a higher service rate. The authors designed multiple algorithms to optimize the resource reservation decisions, or control the service rates in the queues for each job class, to maximize the profit of the cloud. Instead of using separate queues, Khazaei et al. [35] model the entire cloud as an $M/G/n$ queue. The authors assume homogeneous jobs and homogeneous servers in the cloud, so the service rates of all jobs are independent and identically distributed (i.i.d.). With these assumptions, the authors evaluate the queue length distribution and the response time distribution in terms of the job arrival rate, job service rate, and number of servers in the cloud. The theoretical results allow the cloud provider to calculate a number of servers required to satisfy the workload with a desired quality of service (QoS).

In practice, the management system buffer that holds the waiting jobs in cloud has a finite size according to the hardware settings. However, the works above [17, 35] assume the buffer has an infinite length due to its large size. In contrast, some works (e.g., [11]) consider the limitation of the queue length and use $M/G/n/C$ queues. The fourth parameter C in the queue notation denotes the maximum length of the queue, so the new jobs are blocked if the queue already contains C jobs. Chang et al. [11] assume the cloud has homogeneous servers and homogeneous jobs, and model the entire cloud as an $M/G/n/C$ queue. The objective of the work is to evaluate the cloud performance by calculating the queue length distribution, the response time distribution, and the probability of loss. The authors develop an algorithm to approximate the performance parameters, and their numerical results show the high accuracy of the approximation.

From the literature, we observe some common characteristics of the queueing models for solving the server capacity planning problems in clouds. By the nature of

Table 2.3: An overview of the problem assumptions and the solution techniques in the cloud management literature with combinatorial models.

Literature	Problem Assumptions		Solution Techniques		
	Objective	SLA	Solution Model	Load Balancing	Admission Control
[57]	cost minimization	no SLA limitation	integer programming model	class-aware	QoS-agnostic
[51]	operating and transition cost minimization	response time	integer programming model	class-aware	QoS-aware
[24]	minimizing the energy consumption and the response time, maximizing the robustness and the dynamism	response time	mixed-integer programming model	class-aware	QoS-aware
[71]	minimizing the number of operating servers	response time	constraint programming model	class-aware	QoS-aware

queueing models, each queue receives homogeneous jobs and serves them in a first-in-first-out (FIFO) order, which implies the queues cannot support the admission controls that depend on the QoS of the jobs (see Table 2.2). In our work, we classify the jobs based on their QoS requirements, and different queues are used to model the workload of different classes of jobs. Thus, more resources are allocated to the jobs with higher QoS requirements in the scheduling stage. Another observation is that a queueing model provides an accurate evaluation of the performance in a stochastic system, such as a server or a cloud. However, the queueing model cannot provide a detailed job-to-machine allocation strategy for the cloud system, especially when multiple types of jobs can be allocated into the same machine. In the next section, we discuss the works using combinatorial optimization models to find the optimal job-to-machine allocation for the cloud systems.

Combinatorial Optimization

In the scheduling community, combinatorial optimization techniques including mixed integer linear programming and constraint programming are used to solve the scheduling optimization problems for cloud systems [70]. Compared to the queueing models, the combinatorial models can explicitly represent the problems of assigning hetero-

geneous jobs to heterogeneous servers with the minimum cost. In this section, we review works that assume the cloud uses servers with different resource capacities to serve the jobs with different resource requirements, since it fits the assumption in our work (see Section 3.2), and it is a common assumption in the combinatorial optimization works. Table 2.3 shows the problem definitions and the solution models in the short-term resource planning literature with combinatorial models. The table first lists the papers using linear models, such as integer programming and mixed-integer programming models, then the works with CP models are presented. The works with the same solution models are ordered chronologically.

The linear models including linear programming (LP), integer programming (IP), and mixed-integer programming (MIP) models are common techniques for the optimization problems with linear objective and constraints (see Section 2.1.2 for details). Many past works have used the linear models to represent the resource planning problems in the cloud [57, 51, 24], since the operating cost objective is linear in the number of operating machines. Among these works, different constraints are considered to optimize the cost in different clouds. Speitkamp and Bichler [57] formulate an IP model to find the cheapest set of servers to satisfy the varying workload in clouds. They assume the number of customer requests changes in different periods, and the cloud reallocates the jobs in each period while operating the same set of servers in all periods. The authors introduce a set of linear constraints to restrict the job reallocation, such as forcing a set of jobs to stay in the same server and limiting the total number of reallocated jobs. They also introduce multiple heuristics based on the linear relaxation of the IP model to approximate the optimal solution, since they prove the IP problem is NP-hard. The work includes an extensive empirical evaluation on the quality of solutions from different heuristics.

To handle the changes in demands, Sharma et al. [51] reallocate the jobs to satisfy the new workload, and they call the mechanism "migration". Moreover, the authors design their resource planning algorithm to support multiple mechanisms for reconfiguration: local resizing, replication, and migration. The local resizing mechanism resizes the server capacities to fit the new resource requirements of the assigned job. The replication mechanism creates new copies of the jobs with new resource requirements and loads them to the cloud. The migration mechanism is similar to the job reallocation mentioned above, which is reassigning the in-process jobs to new servers. Each mechanism has different sources of cost (see Section III - B in Sharma et al. [51] for details). The experimental studies compare the algorithms using different mechanisms or combined mechanisms, and the results demonstrate the benefits of unifying both replication and migration into a single approach.

Some clouds have more objectives than only minimizing the cost. Guerout et al. [24] develop a multi-objective mixed integer non-linear problem to manage the servers in cloud. The objectives are minimizing the energy consumption and the response time, and maximizing the robustness and the dynamism. They define the robustness as the opposite of the average number of jobs in each server, since losing fewer jobs from a server failure leads to a higher robustness. The dynamism is measured by the amount of extra space in the operating servers that can be used when a workload peak arrives. They construct a MIP model to solve the problem with an objective function as the weighted sum of the multiple objectives. The non-linear constraints are converted to linear constraints by discretizing a continuous variable and introducing a set of binary selection variables.

Constraint programming (CP) is a logic based approach that finds feasible solutions for the constraint satisfaction problems (CSP), or the optimal solution when the problem has an objective to optimize [48]. Compared to the LP problems, the logic-based CP solver can solve the problems with non-linear constraints and objective [48]. Fewer studies have used constraint programming model to solve the resource planning problems in cloud compared to the linear models. Zhang et al. [71] consider the QoS of the jobs by including the performance satisfaction-level of the jobs in the objective. The performance satisfaction-level is a non-linear performance evaluation score based on the actual response time and the desired response time of the job. Thus, the authors introduce a CP model to solve the non-linear optimization problem. They also utilize CP to solve the resource planning problems to minimize the number of operating servers.

Compared to the queueing models, combinatorial models relax the stochastic behaviors of the cloud and focus on the deterministic scheduling decision optimization. To take the advantage of both models, some literature hybridizes the techniques and applies it to the cloud in multiple stages. These works are discussed in the next section.

Hybrid Algorithms

In previous sections, we reviewed the works that use either queueing models or combinatorial models to solve resource planning problems in cloud. However, neither of the models can fully represent the cloud by themselves. Queueing models assume that each queue contains the same service units, where each provides one-to-one services to the jobs. Thus, the queues either assume each machine processes a single job at a time, or each machine contains multiple virtual machines that process the same kind of jobs. Neither of the structures considers the job-to-machine allocation with

multiple types of jobs. In combinatorial optimization models, the allocation problem is well studied, but all combinatorial models solve deterministic problems. Hence, the researchers relax the stochastic behavior in the clouds by using the expected values or assuming a system with deterministic job arrivals and service lengths to apply combinatorial models. Table 2.4 and 2.5 show the problem definitions and the solution techniques of the works that use hybrid algorithms to overcome these difficulties.

Some queueing studies [26, 66, 56] add additional heuristics to optimize the scheduling decisions based on the performance parameters calculated by the queueing models, such as the waiting time distribution and the queue length distribution. Guo et al. [26] construct a load balancing heuristic that tries to maximize the throughput of the cloud with a job-to-machine assignment based on the server performance scores, which is the weighted sum of the mean waiting time, queue length, and utilization rate of each server. Each server has a different service rate, and is modeled by an $M/M/n$ queue. Wang et al. [66] also assume the cloud contains heterogeneous servers with different service rates, but they use an $M/M/n/C$ queue to model the queue for each type of servers. Unlike the $M/M/n$ queue, the $M/M/n/C$ queue rejects new jobs if the queue length reach the limit C (see Section 2.2.2). Moreover, this work considers impatient customers, which means the customer requests are cancelled when their individual maximum waiting time (MWT) is reached. The authors analyze the mean response time and the loss probability of the jobs with a given load balancing strategy on different types of servers. Based on the performance analysis, the authors design a heuristic to approximate the optimal load balancing decision within a reasonable time, where the objective is to maximize the profit in the cloud.

Song et al. [56] use a two-stage management framework to minimize the operating cost of the cloud. In the first stage, the jobs arrive to the cloud, and each job contains multi-class tasks. For example, one job may have computing tasks and display tasks, and another job contains storage tasks only. The authors use an $M/G/n/C$ queue to model the tasks in each class, then build a task management algorithm to calculate the number of virtual machines (VM) required in each queue to satisfy the deadline of the tasks. In the second stage, the framework solves a variant of multi-dimensional bin packing problem to pack the VMs into the physical machines (PMs) with the minimum operating cost. The authors modify four existing heuristics to approximate the optimal VM allocation: user-directed assignment (UDA), Min-Max, Max-Min, and the suffrage heuristic (see Section 4.4 in Song et al. [56] for details), since the multi-dimensional bin packing problem is known to be NP-complete. In the end, the work shows the simulation results of the management framework with different heuristics, and compares their performance under different simulation settings.

In the optimization studies, some metaheuristics are used to solve the optimization problems with less computational complexity. Alharbi et al. [3] formulate the cloud resource planning problem as a constraint satisfaction problem (CSP) with an objective of minimizing the energy consumption. Then they construct an ant colony system (ACS) based algorithm to solve the CSP. The ant colony system [16] is a metaheuristic that searches for the optimal solution by leaving stronger pheromone on the path to the better solution. Thus, the other ants have a higher probability to follow the paths leading to good solutions, and explore around them for an improvement. The authors construct the ACS embedded with a new heuristic that updates the pheromone based on the objective value formulated in the CSP. They show the usefulness of the new heuristic with some empirical results.

Among the reviewed studies, Tran et al. [60] propose the framework that is closest to our approach in Chapter 3. They assume the cloud uses servers with different resource capacities to process the jobs with different resource requirements. The authors introduce a three-stage framework to calculate the optimal scheduling strategy that maximizes the throughput. The first stage of the framework uses queueing theory to formulate the resource requirements of the jobs as a linear expression with their arrival rates, then it calculates the optimal resource allocation plan of the servers to maximize the throughput. In the second stage, the authors generate a set of non-dominated bins for each server to consider the resource loss from imperfect matching of the server capacity and the actual usage. Based on the optimal resource allocation plan in the first stage, each non-dominated bin of a server is a set of the jobs that have a positive number of resources allocated from the server and their total resource requirements is less or equal to the server capacity. Non-dominated indicates that adding any job into the bin will violate the server capacity. Then an LP model is introduced to calculate the optimal job allocation plan to maximize the cloud throughput by assigning the servers to the optimal non-dominated bins. We borrowed the idea of bin generation and renamed it as the configuration generation in Section 3.3.3 of the hybrid framework for the short-term server capacity planning problem. In the final stage, the framework performs online job scheduling based on the solution from the second stage.

Overall, most of the works with hybridized algorithms have a more complete view of the cloud compared to the works with only queueing models or combinatorial models. This observation motivates us to perform hybridization in Chapter 3.

Table 2.4: An overview of the problem assumptions in the cloud management literature with hybrid algorithms.

Literature	Problem Assumptions			
	Objective	Server	Job	SLA
[26]	throughput optimization	heterogeneous servers	homogeneous jobs	no SLA
[66]	profit maximization	heterogeneous servers	heterogeneous jobs with different patience	response time
[56]	cost minimization	homogeneous servers	heterogeneous jobs	response time
[3]	energy consumption minimization	heterogeneous servers	heterogeneous jobs	no SLA
[60]	throughput maximization	heterogeneous servers	heterogeneous jobs	no SLA

Table 2.5: An overview of the solution techniques in the cloud management literature with hybrid algorithms.

Literature	Solution Techniques		
	Solution Model	Load Balancing	Admission Control
[26]	load balancing heuristic based on performance parameters calculated by the M/M/n queues	class-agnostic	QoS-aware
[66]	load balancing heuristic based on performance parameters calculated by the M/M/n/C queues	class-aware	QoS-agnostic
[56]	M/G/n/C queue for VM requirement calculation and heuristics for VM to PM allocation	class-aware	QoS-agnostic
[3]	CSP solved by ACS based algorithm	class-aware	QoS-aware
[60]	hybrid queueing theoretic and combinatorial optimization scheduling algorithm	class-aware	QoS-agnostic

2.2.3 Long-term Resource Planning Problem

In the long-term, the cloud providers need to make periodic server purchasing decisions to update the cloud capacity to satisfy the changing workload. We call the problem of computing the optimal server purchasing plan with the minimum cost the long-term resource planning problems. Unfortunately, none of works have studied the long-term resource planning problems in clouds while considering the price of purchasing servers. Roy et al. [49] optimize the machine leasing plans in multiple periods depending on the workloads, while assuming there are infinite servers ready for leasing.

The inventory control problem in the manufacturing and service industries has goals similar to the long-term resource planning problem. The goal of an inventory control problem is to calculate a periodic order plan for the industries to satisfy their periodic demands. Some of the works have applied linear programming models to find the optimal order plan with the minimum cost [21, 27].

Gans and Zhou [21] studied the demands in call centers, which are typical service industries. The call center has to make hiring plans based on the future demands and the structure of the call center. The authors use linear programming (LP) models to calculate the optimal hiring plan with the minimum cost in the call centers with different structures. They have considered the call centers with or without learning, turnover, and positive training time. The learning effect represents the changes of the employee types as an employee may change their type due to the promotion or the new skills learned. The employees also may quit their positions over time, where the rate of quitting is denoted by the turnover rate, and new employees require a training period before they start to provide services. Clouds have similar structures to the call centers, where the learning, turnover, and training time corresponds the reconfiguration, break down rate, and setting time of the servers, respectively. Thus, their work may be adapted to the cloud environment.

The manufacturing industries have to purchase raw materials for their production, and the products are periodically used to meet demand. Extra products and raw materials must be stored with some cost. Haksever and Moussourakis [27] propose a mixed-integer programming (MIP) model to optimize the orders of heterogeneous raw materials to satisfy the demand of multiple products with the minimum storage cost. Clouds have similar constraints on reserving the resources for the jobs in different classes compared to the constraints on splitting the the heterogeneous materials to multiple products in the manufacturing industries. The authors suggest using MIP models for the inventory control problems with complicated structures, since MIP can handle a large number of constraints. However, the constraints increase the

computational complexity of the MIP model, so the authors implemented a dedicated MIP solver for the inventory control problems. The numerical results show the MIP model and the solver are viable tools for the existing inventory control problem.

2.3 Summary

In this chapter, we introduced the theoretical background of queueing models and combinatorial optimization models. Then we reviewed works on resource planning problems in cloud. The cloud environment has been studied in many approaches, so we grouped the studies based on their problem definitions and solution techniques. By comparing the groups of works, we discuss the different focuses of the different mathematical models.

Chapter 3

Hybrid Framework for Short-term Capacity Planning in Cloud Systems

3.1 Introduction

As the demand for cloud computing increases, cloud service providers require capacity management tools that minimize operating costs while maintaining a promised quality of service. In past studies [64, 69], authors used a variety of queueing models to analyze the performance of cloud systems considering the stochastic arrival times and lifetimes of the arriving jobs. However, typical queueing models do not support processing multiple simultaneous jobs on a single server. In contrast, studies based on combinatorial optimization focus on scheduling strategies for jobs [38, 62], where multiple jobs can be allocated to a single server without exceeding its resource capacity. Yet, since the combinatorial models are deterministic, the stochastic processes in the cloud systems are usually simplified through the use of expected values. Moreover, such point estimates only evaluates the average performance but cannot represent the stochastic operating details of the system, such as the waiting time distribution and the queue length distribution. In our problem, the cloud providers are assumed to maintain a Service Level Agreement (SLA) with their customers to guarantee their jobs will be processed in a short time. Without the response time distribution, it is hard to develop combinatorial-based frameworks that would be able to guarantee SLAs in a stochastic system.

A cloud system typically contains multiple clusters, where each cluster has an independent pool of servers and jobs. Every server pool consists of multiple types of servers, where each type has its own resource capacity and operating cost. On

the demand side, customer requests can be clustered into classes of jobs, where jobs in the same class have the same expected resource requirements, inter-arrival time distributions, and lifetime distributions that may be calculated based on historical data. In this work, we focus on generating an optimal set of servers per cluster according to its workload and a scheduling strategy for the servers that satisfies the SLA of all jobs.

In order to solve this problem that is hard for both pure queueing models and pure combinatorial models, we combine them by using the queueing models to find a deterministic resource requirement satisfying a strict SLA in the stochastic cloud systems, and then pack the resource requirement on servers to form a server requirement. The packing problem is a multi-dimensional bin packing problem, and combinatorial optimization models, such as integer programming (IP) model, are one of the state-of-art approaches for this problem. Thus, the framework is described in two stages:

- Stage I: a queueing model is formulated for each class of jobs. Based on the arrival distribution and lifetime distribution of the jobs, the queueing model calculates the number of "slots" needed to satisfy the SLA of this class, where each slot can only serve one job at a time.

- Stage II: a combinatorial model is constructed to solve a multi-dimensional bin packing problem. The objective is to find the optimal set of servers that have enough resource capacity to pack all the slots required in Stage I with the minimum operating cost.

3.2 Problem Definition

As the operational cost, such as energy cost, in the cloud systems increases, the problem of finding the set of servers that minimizes the operating cost while satisfying the service level agreements (SLA) of all jobs becomes increasingly important. We name this problem the Short-term Server Capacity Optimization Problem (SSC-OP). To solve the SSC-OP, we define a two-stage framework to find an optimal set of servers operating in a cloud system. Moreover, the minimum operating cost solution is used to represent the short-term operating cost in the long-term model studied in Chapter 5.

Table 3.1 contains the parameters used in this chapter. In this short-term framework, the servers used by the cloud system are not all identical, and the jobs may have different resource requirements, inter-arrival time distributions, lifetime distributions, and SLA requirements. We differentiate the servers into different *types* depending on

Table 3.1: Notation

Job Parameters	
I	The set of all job classes.
$\frac{1}{\lambda_i}$	The expected inter-arrival time of jobs in class i, $i \in I$.
$\frac{1}{\mu_i}$	The expected lifetime of jobs in class i, $i \in I$.
c_i	The expected CPU requirement of jobs in class i, $i \in I$.
m_i	The expected memory requirement of jobs in class i, $i \in I$.
(t_i, α_i)	The SLA restricts the probability of a jobs in class i waits for more than t_i must be less than or equal to α_i, $i \in I$.
Server Parameters	
J	The set of all valid server types.
O_j	The operating cost of each server in type j, $j \in J$.
C_j	The CPU capacity of servers in type j, $j \in J$.
M_j	The memory capacity of servers in type j, $j \in J$.
Decision Variables	
z_j	The number of type j servers to operate, $j \in J$.

their CPU capacities, memory capacities, and operating costs. Let J be the set of server types, then for each server with type $j \in J$, it has a CPU capacity of C_j and a memory capacity of M_j. The operating cost of this server is denoted by O_j.

With a set of job *classes* I, we assume the jobs belonging to class $i \in I$ arrive to the cloud system stochastically with inter-arrival times being independent and identically distributed (i.i.d.). The expected inter-arrival time for the jobs in class i is $\frac{1}{\lambda_i}$. The lifetimes of the jobs in each class i are also assumed to be i.i.d. with a mean of $\frac{1}{\mu_i}$. Each class i job requires c_i amount of CPU and m_i amount of memory in expectation.

When jobs arrive to the cloud system, they must be assigned to servers with enough resources or join a queue. We assume that queues have infinite capacities. The waiting times of the jobs in class i must follow the SLA of its class: the probability of a job waits for more than t_i time units (we use seconds in this chapter) should be less than or equal to α_i. The SLA of class i can be defined as,

$$P(W_i > t_i) \leq \alpha_i \tag{3.1}$$

where W_i is the random variable that corresponds to the waiting time of the jobs in class i.

The main challenge is to make a deterministic decision on the number of servers of each type to operate while minimizing the cost of the underlying stochastic system and guaranteeing SLAs. We design this framework to split the stochastic side and the deterministic side of the SSC-OP into two subproblems, and then solve each of them with dedicated models.

3.3 Hybrid Framework for Short-term Server Capacity Optimization

3.3.1 Two Stage Framework with Service Slots

In order to solve the SSC-OP, we propose a framework that combines queueing models and combinatorial models. This framework contains two stages: stage I uses queueing theory to quantify the resource requirements based on the stochastic workload; stage II applies combinatorial optimization to find an optimal server plan.

Queueing theory studies the performance of a system with stochastic processes, such as the waiting queue in call centers or banks [7, 36]. However, most queueing models assume each agent serves one customer at a time, which means the servers in cloud systems cannot be modelled directly: cloud systems allow each server to execute multiple jobs simultaneously provided server capacity is respected. Thus, we introduce the notion of *service slots* to represent the basic agents in queues where each slot processes at most one job at a time. More specifically, each service slot is a unit of capacity reserved in an arbitrary server, which has a resource capacity for one job only. Service slots for different job classes have different sizes, since the jobs in different classes have different resource requirements. For example, suppose that jobs in class i require 4GB of memory and 2 CPU cores in expectation, then the service slot for jobs in class i will be a package of 4GB memory and 2 CPU cores reserved on an arbitrary server. We assume the the jobs in the same class have the same resource requirements that equal to the expectation. The server operating plan produced by the framework introduced later can guarantee the SLAs of all jobs assuming each job can fit into a slot for its class. However, the assumption is not always satisfied in practice. We can set the size of service slot to match the largest resource requirement of the jobs in each class. With this naive strategy, we spend additional resources to guarantee the SLAs of all jobs.

Using service slots, we are able to connect the resource requirement solution from the queueing models to a combinatorial model and obtain the server requirements as the final result. In the second stage, a combinatorial model is used to assign each slot to an operating server. There are two main decisions in the combinatorial model: the set of operating servers and the set of service slots assigned to each server. We will introduce an integer programming (IP) model in Section 3.3.3 to solve this multi-dimensional bin packing problem.

3.3.2 Stage I: Queueing Model

In this stage, we use a queueing model to calculate the number of service slots needed to satisfy specific service level agreements (SLAs). We experiment two formulations of the queueing model: an $M/M/n$ queue and a $GI/GI/n$ queue. Each formulation has different assumptions on the inter-arrival time and lifetime distributions of the jobs (see Section 2.1.1). Recall that in the problem definition of the SSC-OP, we have different SLAs for jobs in different classes. Thus, we build separate queueing models, one per job class, to find the minimum number of service slots for each class. In our work, we apply $M/M/n$ queues and $GI/GI/n$ queues as a starting point, since they both are well-studied queues with known waiting-time distributions. In future work, we can replace stage I by a more sophisticated queueing model, and the framework should not be affected as long as the queueing model produces the minimum service slot requirements to satisfy the SLAs.

Waiting Time Distribution

Recall that the jobs in class i have expected inter-arrival time represented by $\frac{1}{\lambda_i}$, expected lifetime of $\frac{1}{\mu_i}$, and that the SLA restricts the probability of these jobs waiting for more than t_i seconds to less than or equal to α_i, as defined in Equation 3.1.

The SLA can be verified through the waiting time distribution of the queue. Suppose we have a function $PW(\frac{1}{\lambda_i}, \frac{1}{\mu_i}, t_i, n_i)$ which represents the probability that a job waits more than t_i seconds in the queue for class i with n_i service slots. Such functions for $M/M/n$ queue and $GI/GI/n$ queue are presented in Section 2.1.1. Thus, s_i, the minimum number of service slots needed for the class i jobs can be defined as the following,

$$s_i = \min \left\{ n_i \in \mathbb{Z}^+ | PW\left(\frac{1}{\lambda_i}, \frac{1}{\mu_i}, t_i, n_i\right) \leq \alpha_i \right\}.$$

The queueing models calculate the steady state waiting time distributions of the queue. In practice, the jobs arrive and leave the cloud with a high frequency (shown in the dataset experimented in Chapter 4), so the cloud system reaches the steady state in a short time. Moreover, we make the server management decisions while the cloud is operating. Thus, it is reasonable to assume the cloud is in a steady state when we apply the management decisions.

Note the s_i cannot be calculated directly since there are factorials in the waiting time distribution calculations, which do not have inverse functions. Therefore, we use a binary search Algorithm 1 to obtain s_i. The parameters $\frac{1}{\lambda_i}$, $\frac{1}{\mu_i}$, and t_i are constants in the queue of jobs from a single class i, so we use a function $p(n)$ to calculate the

probability of delay exceeding t_i in the queue for class i with n slots, where

$$p(n) = PW\left(\frac{1}{\lambda_i}, \frac{1}{\mu_i}, t_i, n\right).$$

The minimum service slots required by the cloud system $\{s_i\}_{i\in I}$ will be the input to

Algorithm 1 Binary Search for Minimum Service Slot Requirement

```
max ← 1
min ← 0
while p(max) > α_i do
    min ← max
    max ← max * 2
end while
while max − min > 1 do
    mid ← (max + min)/2
    if p(mid) ≤ α_i then
        max ← mid
    else
        min ← mid
    end if
end while
return max
```

the combinatorial mode in the second stage.

3.3.3 Stage II: A Combinatorial Model for Service Slot Packing

Configurations

With the service slot requirements calculated by the queueing models, we aim to find the cheapest set of servers that have sufficient capacity to pack all service slots. This problem is a variant of the VSBPP with variable costs (see Section 2.1.3), where we have two independent weights for each item: CPU and memory requirements. Correspondingly, each server has two independent capacities: CPU and memory capacities. To consider all possible combinations of service slots in each server with two factors, we introduce *server configurations* as vectors describing the number of service slots of each job class in the servers. We define a configuration $\mathbf{v} = (v_1, v_2, ..., v_{|I|}) \in \mathbb{Z}_+^{|I|}$ that contains v_i class i service slots for $i \in I$. Configuration $\mathbf{v}$ is valid for type $j \in J$ servers if and only if $C_j \geq \sum_{i\in I} c_i v_i$ and $M_j \geq \sum_{i\in I} m_i v_i$. For example, suppose there are three classes of jobs, then a configuration $(3, 1, 1)$ contains three class 1 service slots, one class 2 service slot and one class 3 service slots. The total CPU requirement of this configuration is $3c_1 + c_2 + c_3$, and the memory requirement is $3m_1 + m_2 + m_3$, and this configuration is valid only for the servers that have sufficient resource capacity (i.e. $C_j \geq 3c_1 + c_2 + c_3$ and $M_j \geq 3m_1 + m_2 + m_3$).

Servers with the same type have the same resource capacity, so the set of valid server configurations is identical for the same type of servers, which can be defined as,

$$B_j = \left\{ \mathbf{v} = (v_1, ..., v_{|I|}) | \sum_{i \in I} c_i v_i \leq C_j \wedge \sum_{i \in I} m_i v_i \leq M_j \right\}, \quad \forall j \in J$$

where J is the set of server types.

Each server will be assigned to a valid configuration by the combinatorial model that calculates the optimal set of servers. However, each type of servers may have a large set of valid configurations to choose from, which significantly increases the problem size. The goal of the combinatorial model is to use as few servers as possible to pack all the service slots needed, which means a configuration that allows the server to pack more service slots is always preferred. With this observation, we can decrease the size of the combinatorial model by removing the configurations that are less efficient. We define that a server configuration $\bar{\mathbf{v}}$ dominates another configuration $\mathbf{v}$ if and only if $\forall i \in I, \bar{v}_i \geq v_i$. By this definition, we can always assign the server to a dominant configuration to pack more service slots. Thus, the choices of server configurations can be reduced to the set of non-dominated service configurations, which is defined as,

$$\bar{B}_j = \{\bar{\mathbf{v}} | \bar{\mathbf{v}} \in B_j, \forall \mathbf{v} \in B_j, \mathbf{v} \neq \bar{\mathbf{v}} \Rightarrow \neg(\forall i \in I, v_i \geq \bar{v}_i)\}. \quad \forall j \in J$$

Integer Programming Model

There are many types of combinatorial models available for the packing problems in this stage, including integer programming (IP) model, constraint programming (CP) model, satisfiability (SAT), and some dedicated packing problem solvers [23, 47, 55]. The goal of the packing problem is to find the number of servers and their configurations to satisfy the service slot requirements with the minimum operating cost. Specifically, each server contributes a number of service slots depending on its configuration, and the summation of the service slots from all servers must exceed the service slot requirement. These constraints are naturally expressed as linear inequalities, so it is easy to construct an IP model for this problem. Based on some past studies [2, 72], IP models are strong on reasoning about the linear relationship across all constraints, the strength of CP models comes with their specific structured global constraints, and SAT performs better on the problems with binary decisions. Therefore, we choose to use an IP model (see Section 2.1.2) in this chapter as a starting point, and leave the CP model with specialized global constraints as future works.

Consider the problem in this stage: the IP model should decide on the number of operating servers from each type and their configurations, so the decision variable are as follows,

Decision Variables:

z_j The number of operating servers of type j, $\forall j \in J$

$x_{j\mathbf{v}}$ The number of servers in type j that operate in configuration $\mathbf{v}$. $\forall j \in J, \mathbf{v} \in \bar{B}_j$

With the decision variables, the full IP model is as follows.

Integer Programming Model:

$$\min \sum_{j \in J} O_j z_j \tag{3.2}$$

$$\text{s.t.} \sum_{j \in J} \sum_{\mathbf{v} \in \bar{B}_j} v_i x_{j\mathbf{v}} \geq s_i, \qquad \forall i \in I \tag{3.3}$$

$$\sum_{\mathbf{v} \in \bar{B}_j} x_{j\mathbf{v}} = z_j, \qquad \forall j \in J \tag{3.4}$$

$$z_j \in \mathbb{Z}_+, \qquad \forall j \in J \tag{3.5}$$

$$x_{j\mathbf{v}} \in \mathbb{Z}_+. \qquad \forall j \in J, \mathbf{v} \in \bar{B}_j \tag{3.6}$$

In this IP model, constraints 3.3 enforce that all service slots required by each class of jobs are packed into some server. Constraints 3.4 make sure that each server is assigned to at most one configuration. The objective of this IP model is to minimize the total operating cost of the servers while satisfying other constraints.

Constraints 3.5 and 3.6 are domain constraints for the decision variables. Constraints 3.5 force the number of servers to be an integer. The numbers of servers that are assigned to each configuration are also required to be integers, which is enforced by the set of constraints 3.6.

Linear Relaxation

For the IP models, the computational complexity of the model is in the worst-case exponential in the number of integer variables. In the model above, we used $|J| + \sum_{j=1}^{J} |\bar{B}_j|$ integer variables. According to the definition of the set of non-dominated configurations $\{\bar{B}_j\}_{j \in J}$, the cardinality may grow exponentially with respect to the number of job classes. Thus, as the number of job classes increases, it will become

impossible to solve the model within a reasonable amount of time. In the experiment, we show the run-time of the IP model grows quickly as the problem size increases.

In order to reduce the computational complexity of the IP model, we relax the integer variables z_j and $x_{j\mathbf{v}}$ to continuous variables $\bar{z}_j$ and $\bar{x}_{j\mathbf{v}}$, so the IP model becomes a linear programming (LP) model.

Linear Programming Model:

$$\min \sum_{j\in J} O_j \bar{z}_j \tag{3.7}$$

$$\text{s.t.} \sum_{j\in J} \sum_{\mathbf{v}\in \bar{B}_j} v_i \bar{x}_{j\mathbf{v}} \geq s_i, \qquad \forall i \in I \tag{3.8}$$

$$\sum_{\mathbf{v}\in \bar{B}_j} \bar{x}_{j\mathbf{v}} = \bar{z}_j, \qquad \forall j \in J \tag{3.9}$$

$$\bar{z}_j \in \mathbb{R}_+, \qquad \forall j \in J \tag{3.10}$$

$$\bar{x}_{j\mathbf{v}} \in \mathbb{R}_+, \qquad \forall j \in J, \mathbf{v} \in \bar{B}_j \tag{3.11}$$

Consider the optimal solution given by the linear relaxation $\{\bar{z}_j^*, \bar{x}_{j\mathbf{v}}^*\}_{j\in J, \mathbf{v}\in \bar{B}_j}$. This optimal solution of the relaxation cannot be used as a capacity plan in the cloud system, since we cannot operate a half of server with half of its cost. In order to make the solution valid, we need a rounding strategy for the relaxed solutions. We implemented two different rounding strategies: Rounding by Ordering and Rounding by LP Remodeling.

Rounding by Ordering

In this rounding strategy, we try to make the least change on the number of operating servers suggested by the linear relaxation. First, we round up the number of servers that should be operating, and the rounded solution becomes

$$\hat{z}_j^* = \lceil \bar{z}_j^* \rceil. \quad \forall j \in J$$

Suppose we round down all of the configuration variables $\{\bar{x}_{j\mathbf{v}}^*\}$, then for the servers in each type j, there are $q_j = \hat{z}_j^* - \sum_{\mathbf{v}\in \bar{B}_j} \lfloor \bar{x}_{j\mathbf{v}}^* \rfloor$ servers that do not have an assigned configuration. To decide the configurations for those servers, we sort the variables $\{\bar{x}_{j\mathbf{v}}^*\}_{\mathbf{v}\in \bar{B}_j}$ by their fractional value $(\bar{x}_{j\mathbf{v}}^* - \lfloor \bar{x}_{j\mathbf{v}}^* \rfloor)$ in non-decreasing order and with the variables with the same fractional value ordered arbitrarily. Suppose $\delta_j(\bar{x}_{j\mathbf{v}}^*)$ represents the index of the variable $\bar{x}_{j\mathbf{v}}^*$ in such order, then the variables in set $\{\bar{x}_{j\mathbf{v}}^* | \mathbf{v} \in \bar{B}_j, \delta_j(\bar{x}_{j\mathbf{v}}^*) \leq q_j\}$ are rounded up, and the rest of the configuration variables $\{\bar{x}_{j\mathbf{v}}^* | \mathbf{v} \in$

$\bar{B}_j, \delta_j(\bar{x}^*_{j\mathbf{v}}) > q_j\}$ are rounded down. Notice if the number of servers without a configuration is greater than the total number of configurations (i.e. $q_j > |\bar{B}_j|$), then assigning these servers one-to-one to configurations cannot guarantee a configuration for all servers. In Theorem 3.2, we prove that this will never happen in the LP solution.

Lemma 3.1. $\lceil \sum_{k=1}^{K} n_k \rceil \leq \sum_{k=1}^{K} \lceil n_k \rceil, \forall K \in \mathbb{Z}^+, \forall k \in \{1, 2, ..., K\}, n_k \in \mathbb{R}.$

Proof. Note this lemma is well known (e.g., [41]), we present the proof here for completeness. The lemma is proved by induction.
Base Case: $k = 1$

$$\lceil n_1 \rceil = \lceil n_1 \rceil$$

Induction Step: Assume $\lceil \sum_{k=1}^{K} n_k \rceil \leq \sum_{k=1}^{K} \lceil n_k \rceil, K \in \mathbb{Z}^+$.
Consider the summation with an additional item $n_{k+1} \in \mathbb{R}$, $\sum_{k=1}^{K+1} n_k$.
Let $m_1 = \sum_{k=1}^{K} n_k - \lfloor \sum_{k=1}^{K} n_k \rfloor$, $m_2 = n_{K+1} - \lfloor n_{K+1} \rfloor$.
Notice $0 \leq m_1, m_2 < 1$, if $m_1 = m_2 = 0$,

$$\begin{aligned} \lceil \sum_{k=1}^{K} n_k \rceil + \lceil n_{K+1} \rceil &= \sum_{k=1}^{K} n_k + n_{K+1} \\ &= \lceil \sum_{k=1}^{K+1} n_k \rceil. \end{aligned}$$

If $m_1 = 0, m_2 > 0$ or $m_1 > 0, m_2 = 0$,

$$\begin{aligned} \lceil \sum_{k=1}^{K} n_k \rceil + \lceil n_{K+1} \rceil &= \lfloor \sum_{k=1}^{K} n_k \rfloor + \lfloor n_{K+1} \rfloor + 1 \\ &= \lfloor \sum_{k=1}^{K} n_k \rfloor + \lfloor n_{K+1} \rfloor + \lceil m_1 + m_2 \rceil \\ &= \lceil \sum_{k=1}^{K+1} n_k \rceil. \end{aligned}$$

If $m_1 > 0, m_2 > 0$ and $m_1 + m_2 \leq 1$,

$$\begin{aligned}\lceil\sum_{k=1}^{K} n_k\rceil + \lceil n_{K+1}\rceil &= \lfloor\sum_{k=1}^{K} n_k\rfloor + 1 + \lfloor n_{K+1}\rfloor + 1 \\ &= \lfloor\sum_{k=1}^{K} n_k\rfloor + \lfloor n_{K+1}\rfloor + 2 \\ &> \lfloor\sum_{k=1}^{K} n_k\rfloor + \lfloor n_{K+1}\rfloor + \lceil m_1 + m_2\rceil \\ &= \lceil\sum_{k=1}^{K+1} n_k\rceil.\end{aligned}$$

If $m_1 > 0, m_2 > 0$ and $1 < m_1 + m_2 \leq 2$,

$$\begin{aligned}\lceil\sum_{k=1}^{K} n_k\rceil + \lceil n_{K+1}\rceil &= \lfloor\sum_{k=1}^{K} n_k\rfloor + 1 + \lfloor n_{K+1}\rfloor + 1 \\ &= \lfloor\sum_{k=1}^{K} n_k\rfloor + \lfloor n_{K+1}\rfloor + 2 \\ &= \lfloor\sum_{k=1}^{K} n_k\rfloor + \lfloor n_{K+1}\rfloor + \lceil m_1 + m_2\rceil \\ &= \lceil\sum_{k=1}^{K+1} n_k\rceil.\end{aligned}$$

Thus, in all cases, $\lceil\sum_{k=1}^{K} n_k\rceil + \lceil n_{K+1}\rceil \geq \lceil\sum_{k=1}^{K+1} n_k\rceil$. According to the assumption $\lceil\sum_{k=1}^{K} n_k\rceil \leq \sum_{k=1}^{K}\lceil n_k\rceil$,

$$\lceil\sum_{k=1}^{K+1} n_k\rceil \leq \lceil\sum_{k=1}^{K} n_k\rceil + \lceil n_{K+1}\rceil \leq \sum_{k=1}^{K+1}\lceil n_k\rceil.$$

□

Theorem 3.2. *For any server type j, the number of unassigned servers q_j is always less than or equal to the total number of non-dominated configurations $|\bar{B}_j|$.*

Proof. Consider an arbitrary server type $j \in J$. By definition,

$$\begin{aligned}q_j &= \hat{z}_j^* - \sum_{\mathbf{v}\in\bar{B}_j} \lfloor\bar{x}_{j\mathbf{v}}^*\rfloor \\ &= \lceil\sum_{\mathbf{v}\in\bar{B}_j} \bar{x}_{j\mathbf{v}}^*\rceil - \sum_{\mathbf{v}\in\bar{B}_j} \lfloor\bar{x}_{j\mathbf{v}}^*\rfloor.\end{aligned}$$

By Lemma 3.1

$$\begin{aligned}
&\leq \sum_{\mathbf{v}\in\bar{B}_j} (\lceil \bar{x}^*_{j\mathbf{v}} \rceil - \lfloor \bar{x}^*_{j\mathbf{v}} \rfloor) \\
&\leq \sum_{\mathbf{v}\in\bar{B}_j} (\lfloor \bar{x}^*_{j\mathbf{v}} + 1 \rfloor - \lfloor \bar{x}^*_{j\mathbf{v}} \rfloor) \\
&= \sum_{\mathbf{v}\in\bar{B}_j} 1 \\
&= |\bar{B}_j|.
\end{aligned}$$

□

However, with this rounding strategy, the rounded solution is not guaranteed to be feasible for the original IP problem, since rounding down some of the configuration variables might violate the service slot requirements. According to the experiment in a previous study in a similar context [60], the gap between the provided slots and the demanded slots is insignificant compared to the total slot requirements, since most servers are properly configured by the relaxed solution when the number of operating servers is large.

Now we consider the additional cost caused by the rounded linear relaxation solution compared to the solution from the original IP model. Suppose the optimal solution of the original IP model is $\{z^*_j, x^*_{j\mathbf{v}}\}_{j\in J, \mathbf{v}\in\bar{B}_j}$. Notice that this solution is feasible for the linear relaxation, which means,

$$\sum_{j\in J} O_j \bar{z}^*_j \leq \sum_{j\in J} O_j z^*_j. \tag{3.12}$$

Thus, the extra cost of the rounded linear relaxation solution is an upper bound of the total operating cost of each server type.

Theorem 3.3. *The cost difference between the linear relaxation solution and the optimal integer solution is bounded by the total operating cost of each server type, i.e.*

$$\sum_{j\in J} O_j \hat{z}^*_j - \sum_{j\in J} O_j z^*_j \leq \sum_{j\in J} O_j.$$

Proof.

According to Equation 3.12

$$\begin{aligned}\sum_{j\in J} O_j \hat{z}_j^* - \sum_{j\in J} O_j z_j^* &\leq \sum_{j\in J} O_j \hat{z}_j^* - \sum_{j\in J} O_j \bar{z}_j^* \\ &= \sum_{j\in J} O_j \lceil \bar{z}_j^* \rceil - \sum_{j\in J} O_j \bar{z}_j^* \\ &\leq \sum_{j\in J} O_j (\lceil \bar{z}_j^* \rceil - \bar{z}_j^*) \\ &\leq \sum_{j\in J} O_j.\end{aligned}$$

□

As the number of operating servers increases, the operating cost calculated by the LP model will get relatively closer to the operating cost from the IP model. Therefore, the LP model can produce reasonable approximations for the optimal server capacity plans in large cloud systems.

Rounding by IP Remodeling

As noted, the previous rounding strategy does not guarantee the service slot requirements are satisfied. Thus, we introduce a new rounding strategy for the continuous solutions that guarantees satisfaction of the service slot requirements.

Consider the optimal solution given by the linear relaxation $\{\bar{z}_j^*, \bar{x}_{j\mathbf{v}}^*\}_{j\in J, \mathbf{v}\in \bar{B}_j}$. According to our experiments, the number of configurations that are used in the solution are much smaller than the number of all non-dominated configurations in every server *type*. Based on this observation, we re-solve the problem with the subset of used server types $\hat{J}$ and the subsets of used configurations $\hat{B}_j$ in each class $j \in \hat{J}$.

$$\hat{J} = \{j \in J | \bar{z}_j^* > 0\}, \tag{3.13}$$

$$\hat{B}_j = \{\mathbf{v} \in \bar{B}_j | \bar{x}_{j\mathbf{v}}^* > 0\}. \quad \forall j \in \hat{J} \tag{3.14}$$

Then we can reformulate the IP model as

$$\min \sum_{j\in\hat{J}} O_j \hat{z}_j \tag{3.15}$$

$$\text{s.t.} \sum_{j\in\hat{J}} \sum_{\mathbf{v}\in\hat{B}_j} v_i \hat{x}_{j\mathbf{v}} \geq s_i, \qquad \forall i \in I \tag{3.16}$$

$$\sum_{\mathbf{v}\in\hat{B}_j} \hat{x}_{j\mathbf{v}} = \hat{z}_j, \qquad \forall j \in \hat{J} \tag{3.17}$$

$$\hat{z}_j \in \mathbb{Z}_+, \qquad \forall j \in \hat{J} \tag{3.18}$$

$$\hat{x}_{j\mathbf{v}} \in \mathbb{Z}_+, \qquad \forall j \in \hat{J}, \mathbf{v} \in \hat{B}_j \tag{3.19}$$

This new IP model can be considered as the model with additional constraints that set the variables $\{z_j | j \in J \backslash \hat{J}\}$ and $\{x_{j\mathbf{v}} | j \in \hat{J}, \mathbf{v} \in \bar{B}_j \backslash \hat{B}_j\}$ to be zeros compared to the original IP model. Notice the new IP model is guaranteed to be feasible, which can be proved as the follows.

Theorem 3.4. *The reduced IP model (3.15)-(3.19) has a feasible solution with any $\hat{J}$ and $\hat{B}_j$, such that*

$$\hat{J} = \{j \in J | \bar{z}_j^* > 0\}, \tag{3.20}$$

$$\hat{B}_j = \{\boldsymbol{v} \in \bar{B}_j | \bar{x}_{j\boldsymbol{v}}^* > 0\}, \quad \forall j \in \hat{J} \tag{3.21}$$

where $\{\bar{z}_j^*, \bar{x}_{j\boldsymbol{v}}^*\}_{j\in J, \boldsymbol{v}\in\bar{B}_j}$ is an optimal solution of the LP model (3.7)-(3.11).

Proof. Consider an optimal solution $\{\bar{z}_j^*, \bar{x}_{j\mathbf{v}}^*\}_{j\in J, \mathbf{v}\in\bar{B}_j}$ of the LP model (3.7)-(3.11). We know that for all job class $i \in I$, there exist a type of servers $j^i \in J$ that use a configuration $\mathbf{v}^i$ with $v_i^i > 0$ in the optimal solution, which is

$$\forall i \in I, \exists j^i \in J, \mathbf{v}^i \in \bar{B}_{j^i}, s.t. \;\; v_i^i > 0 \wedge \bar{x}_{j^i\mathbf{v}^i}^* > 0.$$

Otherwise, the optimal solution cannot provide any service slot to some job classes, so the solution is no longer feasible, which contradicts our assumption. Then by Equations 3.20 and 3.21, we know $j^i \in \hat{J}$ and $\mathbf{v}^i \in \hat{B}_{j^i}$. Thus, we can construct a feasible solution $\{\hat{z}_j, \hat{x}_{j\mathbf{v}}\}_{j\in\hat{J}, \mathbf{v}\in\hat{B}_j}$ for the reduced IP model (3.15)-(3.19), where

$$\hat{x}_{j\mathbf{v}} = \begin{cases} s_i, & \textit{If } \exists i \in I, \mathbf{v} = \mathbf{v}^i \\ 0, & \textit{Otherwise} \end{cases}, \quad \forall j \in \hat{J}, \mathbf{v} \in \hat{B}_j \tag{3.22}$$

$$\hat{z}_j = \sum_{\mathbf{v}\in\hat{B}_j} \hat{x}_{j\mathbf{v}}. \quad \forall j \in \hat{J} \tag{3.23}$$

To verify the solution is feasible, we check all the constraints in the reduced IP model (3.16)-(3.19). By the definition of configurations, we know $\mathbf{v}^i \in \mathbb{Z}_+^{|I|}$ for all $i \in I$, so $v_i^i > 0 \Rightarrow v_i^i \geq 1$. Thus, we have

$$\begin{aligned}\sum_{j\in\hat{J}} \sum_{\mathbf{v}\in\hat{B}_j} v_i \hat{x}_{j\mathbf{v}} &\geq v_i^i \cdot \hat{x}_{j^i\mathbf{v}^i} \\ &\geq 1 \cdot \hat{x}_{j^i\mathbf{v}^i} \\ &\geq s_i, \quad \forall i \in I\end{aligned}$$

so the set of constraints (3.16) is satisfied. The set of constraints (3.17) is directly implied by Equation 3.23. The domain constraints (3.18)-(3.19) are satisfied since the solution only contains positive integers. Therefore, $\{\hat{z}_j, \hat{x}_{j\mathbf{v}}\}_{j\in\hat{J},\mathbf{v}\in\hat{B}_j}$ is a feasible solution for the reduced IP model (3.15)-(3.19). □

Thus, there exist the optimal solution $\{\hat{z}_j^*, \hat{x}_{j\mathbf{v}}^*\}_{j\in\hat{J},\mathbf{v}\in\hat{B}_j}$, which is always a feasible solution for the original IP model with zeros on all of the undefined variables. Therefore, $\{\hat{z}_j^*, \hat{x}_{j\mathbf{v}}^*\}_{j\in\hat{J},\mathbf{v}\in\hat{B}_j}$ guarantees the satisfaction of the service slot requirements, and its objective value upper bounds the optimal objective value of the original IP.

3.4 Experimental Results

We perform multiple experiments for different parts of our framework. We first run a set of simulations that mimic the job arrivals and service events in cloud systems with different assumptions on their distributions. We compared the quality of the solutions from $M/M/n$ models and $GI/GI/n$ models in different simulations to find the best model in each case. Secondly, we want to evaluate the run-time of the combinatorial models with different problem sizes, and the accuracy of the rounded LP solution as an approximation. We therefore compare the LP solutions with two rounding strategies: ordering and IP remodeling (Section 3.3.3). According to the experimental results, compared to the ordering strategies, the LP model with the IP remodeling strategy produces a better solution in all problem instances while satisfying the service slot requirements. With this observation, we apply the two combinatorial models: the IP model and the LP model with IP remodeling rounding strategy, to the problems with different sizes. In the end, we compare the run-time of the full framework with different models. Moreover, since the approximation errors from the $GI/GI/n$ queueing model and the rounded LP combinatorial model both affect the solution of the framework, then we want to observe the performance of the solution with this combined error. Thus, we test the solution produced by the full

framework on a simulator that mimics the cloud system with multiple job classes.

3.4.1 Queueing Model Experiments

We have introduced two types of queueing models to represent the cloud system: $M/M/n$ queues and $GI/GI/n$ queues. To understand the performance of the queueing model solutions in different types of queue, we implemented two types of simulators that match the assumptions of $M/M/n$ queues and $GI/GI/n$ queues. Each simulator simulates the queue of jobs in one class, since our framework uses one queueing model per job class to calculate its service slot requirement. The simulator assigns the jobs to a number of service slots based on the solution from the queueing model. The inter-arrival times and lifetimes of the jobs in the $M/M/n$ simulators are generated from exponential distributions. In contrast, $GI/GI/n$ simulators use lognormal distributions to generate the inter-arrival times and lifetimes of the jobs.

A set of queue instances are generated to test the queueing models under different parameters: inter-arrival time distribution, lifetime distribution, and Service Level Agreement (SLA). We want to verify the quality of the $M/M/n$ solution and the $GI/GI/n$ approximation in different environments by observing the job waiting time in the simulations. In the queue instances, we set the expected inter-arrival time $\frac{1}{\lambda} = \frac{1}{2}$ second, with different expected lifetime $\frac{1}{\mu} \in \{40, 80, 120, 160\}$ seconds. The standard deviations of the inter-arrival time and the lifetime are $sd \times \frac{1}{\lambda}$ and $sd \times \frac{1}{\mu}$ respectively, where $sd \in \{2, 4, 8\}$. The graphs from these different settings have similar behavior, so we present the graphs with 80 seconds expected lifetime and sd equal to 4. The other figures are presented in Appendix B. With these settings, as the SLA changes, the number of service slots required by the $M/M/n$ models and $GI/GI/n$ models changes.

In Figure 3.1, we compare the service slot requirements calculated by different models as the SLA changes. The red line and the blue line represent the service slot requirements calculated by the $M/M/n$ model and the $GI/GI/n$ model respectively. However, the solution of the $GI/GI/n$ model approximates the minimum number of service slots required for a desired SLA (see Section 2.1.1), so we use a simulation method that finds a solution by increasing the number of service slots in the $GI/GI/n$ simulator until the SLA is satisfied. The green line depicts the solution obtained from the $GI/GI/n$ simulation method. This solution is considered as the minimum service slot requirement for the $GI/GI/n$ queue, since Figure 3.3 shows the performance of the simulation with the solution is close to the SLA limit. The grey dashed line in Figure 3.1 represents the offered load, which is $\frac{\lambda}{\mu}$. If the number of service slots for a class is less than the offered load, then the queue size will keep increasing as

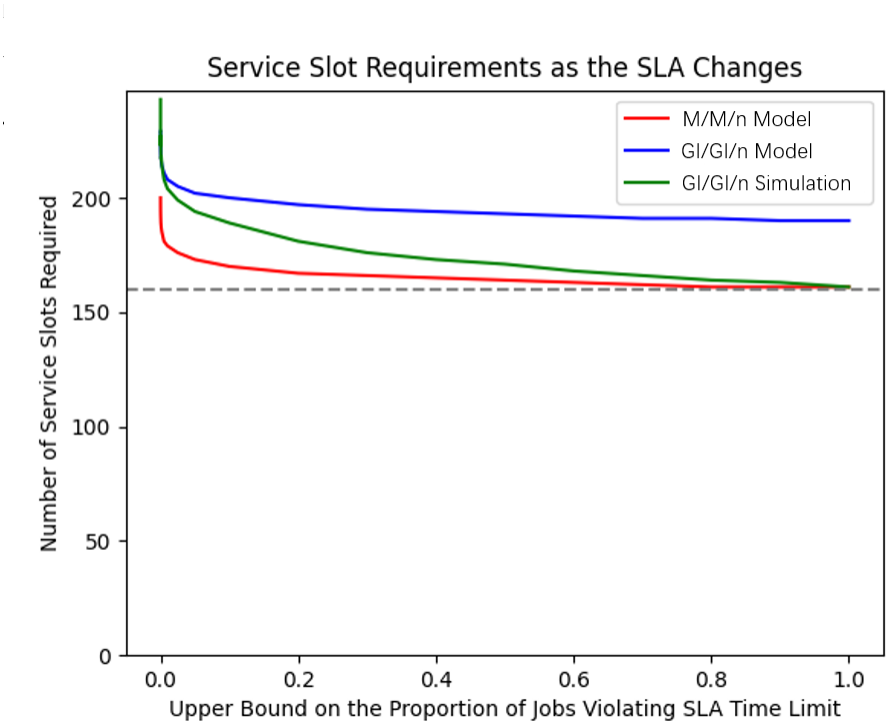


Figure 3.1: The number of service slots required by different models as the upper bound on the probability of a job waits for more than 10 seconds changes.

the system operates, leading to an unstable system. With this lower bound on the number of service slots required by the system, the $M/M/n$ model solution seems to be reasonable since it approaches the offered load as the SLA becoming less restricted, and it grows exponentially when the probability of delay approaches zero. However, the solution of the $GI/GI/n$ model does not approach the offered load as the SLA becoming less restricted and always over estimates the number of service slots required. This is caused by the approximation error from the $GI/GI/n$ model. Recall Equation 2.1

$$PW_{M/M/n}\left(\frac{1}{\lambda}, \frac{1}{\mu}, t, n\right) = \gamma(\rho, n)e^{-(n-\rho)\mu t},$$

and Equation 2.4

$$PW_{GI/GI/n}\left(\frac{1}{\lambda}, \frac{1}{\mu}, t, n\right) \approx \frac{1}{\mu}\gamma(\rho, n)\exp\left(-(n-\rho)\mu t \cdot \frac{2}{C(A)^2 + C(L)^2}\right).$$

Notice the waiting time distribution approximation of the $GI/GI/n$ system (Equation 2.4) contains an additional factor $\frac{1}{\mu}$ (mean lifetime) compared to the waiting time distribution of the $M/M/n$ (Equation 2.1). Thus, with a large lifetime (80 in Figure 3.1), the $GI/GI/n$ model always approximates a larger probability of delay compared

Figure 3.2: The probability of a job waiting for more than 10 seconds in the $M/M/n$ environment system with number of service slots calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes.

to the $M/M/n$ model while other parameters are kept the same. Based on the observation in Figure 3.1, as the SLA becomes more restricted, the $GI/GI/n$ model approximation approaches the result from the $GI/GI/n$ simulation method, which suggests the accuracy of the approximation is increasing as the system becomes more restricted.

Figure 3.2 shows the performance of using the solutions from the $M/M/n$ model and the $GI/GI/n$ model in the simulated cloud system with $M/M/n$ environment. The simulations run for 1,000,000 seconds, so each simulation is expected to have 2,000,000 jobs. The inter-arrival times and lifetimes of the jobs are obtained from exponential distributions in an $M/M/n$ environment. The red line and the blue line in Figure 3.2 depict the probability of a job waits for more than 10 seconds in the simulations that have their number of service slots calculated by the $M/M/n$ model and the $GI/GI/n$ model respectively. The grey line shows the maximum allowance on the probability of more than 10 seconds delay by the SLA. Thus, if the probability of delay observed in the simulation is above the grey line, then the SLA is violated in that simulation. As the graph shows, both models find a number of service slots that satisfies the SLA in all simulations. The solution from the $M/M/n$ model is preferred for the $M/M/n$ system since it is closer to the minimum number of service slots that

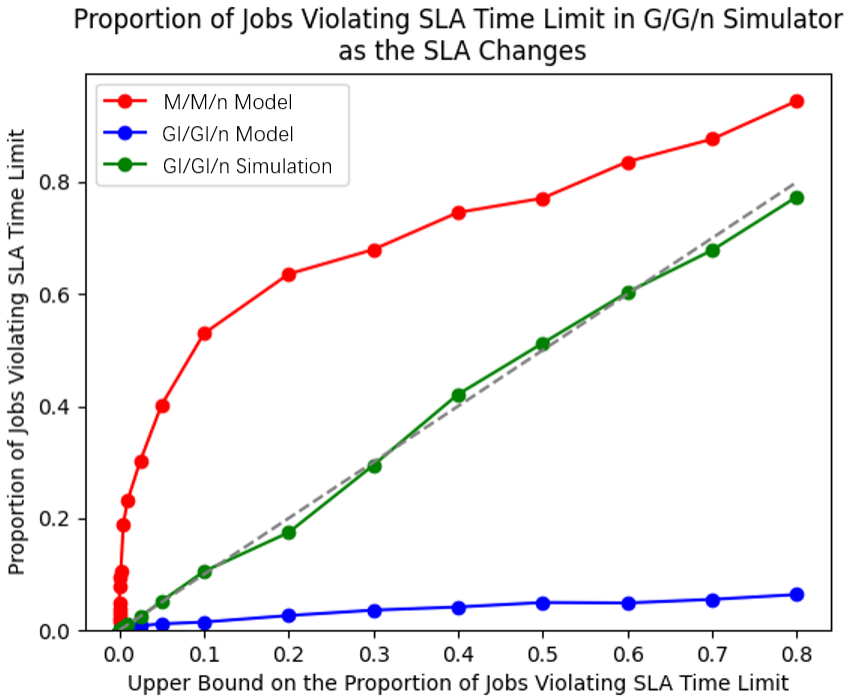


Figure 3.3: The probability of a job waiting for more than 10 seconds in the $GI/GI/n$ environment with the number of service slots calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes.

satisfies the SLA compared to the solution from the $GI/GI/n$ model.

Figure 3.3 shows the performance of the solutions from the $M/M/n$ model and the $GI/GI/n$ model in the simulated cloud system with $GI/GI/n$ environment. The simulations run for 1,000,000 seconds, with around 2,000,000 jobs. The inter-arrival times and lifetimes of the jobs are generated from lognormal distributions in a $GI/GI/n$ environment. The red line, the blue line, and the green line in Figure 3.2 depict the probability of a job waits for more than 10 seconds in the simulations that have their number of service slots calculated by the $M/M/n$ model, the $GI/GI/n$ model, and the $GI/GI/n$ simulation method respectively. The grey line shows the maximum allowance on the probability of more than 10 seconds delay by the SLA, so the simulations with their probability of delay above the grey line violate the SLA. In the $GI/GI/n$ system, the solutions calculated by the $M/M/n$ model cannot guarantee the SLA in any of the experimental conditions, since the standard deviations of the inter-arrival times and the lifetimes are greater than the standard deviations in exponential distributions, which is expected by the $M/M/n$ model. As expected, the simulation with the solution calculated by the $GI/GI/n$ simulation method has a performance closest to the SLA limit, since this method is based on the same simulation as the one in the experiment to obtain the solution. However, the simulation

Table 3.2: Cost Comparison on Different Rounding Strategies

Problem Sizes	Ordering		IP Remodelling		
($\|I\|$, $\|J\|$)	Optimality Gap	Feasibility	Optimality Gap	Feasibility	Run-time
(20, 20)	0.0078%	No	0.0078%	Yes	3.156ms
(20, 30)	0.0084%	Yes	0.0084%	Yes	1.503ms
(20, 40)	0.0364%	Yes	**0.0189%**	Yes	3.636ms
(20, 50)	0.0307%	No	**0.0087%**	Yes	3.861ms
(20, 60)	0.0125%	No	0.0125%	Yes	1.719ms
(30, 20)	0.0047%	No	0.0047%	Yes	2.937ms
(40, 20)	**0.0047%**	No	0.0143%	Yes	17.423ms

method requires an adjustment to the simulation to be applicable on the queue with different inter-arrival time and lifetime distributions. Fortunately, the solutions approximated by the $GI/GI/n$ model always satisfy the SLA in the $GI/GI/n$ system, and the approximation error reduces as the SLA becomes more restricted.

With these observations, we conclude the $M/M/n$ model outperforms the $GI/GI/n$ model in the cloud systems with the inter-arrival time and lifetime of jobs follow exponential distributions, which is the assumption of the $M/M/n$ queues. When the inter-arrival time and lifetime of jobs in the cloud system follow some other distributions, the $GI/GI/n$ model would produce a better solution, and this solution approaches to the minimum service slot requirement as the SLA becomes more restricted.

3.4.2 LP Relaxation with Different Rounding Strategies

In Table 3.2 and Table 3.3, we experimented with the combinatorial models on the same set of problem instances. The set contains 5 problem instances in each problem size, where the numbers of job classes ($|I|$) and the number of server types ($|J|$) are shown in the "($|I|, |J|$)" column. The number of service slots required by each class is set to be 800, and each server has enough capacity for dozens to a hundred jobs. We used C++ to generate the configurations of all server types, with a 3GB upper bound on the memory required by the configurations. The LP and IP models were solved by Gurobi 9.5.1. with a time limit of 300 seconds

We introduced two rounding strategies for the LP model in Section 3.3.3. Table 3.2 shows the comparison of the two strategies in our experiments. The optimality gap is calculated by comparing the rounded solution with the optimal solution of the linear relaxation, since the linear relaxation provides a lower bound on the optimal cost of the IP model (3.2)-(3.6).

$$\text{Optimality Gap} = \frac{\text{Rounded Cost - Linear Relaxation Cost}}{\text{Linear Relaxation Cost}}$$

The "Feasibility" column in Table 3.2 indicates whether the rounded solution is feasible for the IP model (3.2)-(3.6) or not, and the "Run-time" column corresponds to the time spent solving the reduced IP model for rounding.

According to these results, reconstructing the reduced IP model and solving it takes an insignificant amount of time, and the solution cost is very close to the theoretical lower bound of the original IP model (Optimality Gap $< 0.02\%$). In contrast, the LP solution rounded by ordering has a larger cost and cannot guarantee the service slot requirements. Therefore, we chose the IP remodeling strategy for the LP models in later experiments.

3.4.3 Combinatorial Model Experiments

The two types of the combinatorial models we use in the experiments are the IP model and the LP model with rounding. We are interested in the different run-times of the two models and the optimality gap of their solutions compared to the optimal solution. Based on the observations from Table 3.2, we decided to use IP remodelling as the rounding strategy for the LP model, since it calculates a lower cost within a short time.

Table 3.3 shows the average results of running the experiment with the problem instances with different sizes. The "Problem Sizes" section shows the number of classes ($|I|$) and the number of types ($|J|$) in the problem instances. With each problem size, we generate 10 problem instances. Then the average time spent on generating the non-dominated configurations of all servers is presented in "Config Time", and the average number of non-dominated configurations is shown in "Config Size". The "Reduced Config Size" shows the average number of configurations used in the LP optimal solution and the reduced IP model. In the experimental results of each model, "Run-time" is the average time taken to solve the combinatorial model. The run-time of the LP model with IP remodeling is separated into two parts: the run-time of the LP model ("LP Run-time") and the run-time of solving the reduced IP model ("IP Run-time"). The optimality gap is calculated with respect to the optimal solution of the IP model, that is Optimality Gap $= \frac{\text{reduced IP Solution}-\text{IP Solution}}{\text{IP Solution}}$, since the IP finds an optimal solution in all instances.

Similar to the observations in Table 3.2, Table 3.3 shows the reduced IP model takes an insignificant amount of time compared to the run-time of the LP model. This is not surprising by considering the great reduction in the number of configurations (millions to hundreds) for the reduced IP model. In all problem instances, the LP model with the reduced IP model uses less time than the pure IP model, and finds feasible solutions that are very close to the optimal (optimality gap $\leq 0.011\%$) in all

Table 3.3: Run-time and Cost Comparison on Combinatorial Models with Different Problem Sizes

Problem Sizes	IP Model			LP Model with reduced IP model				
($\|I\|$, $\|J\|$)	Config Time	Config Size	Run-time	Reduced Config Size	Run-time	LP Run-time	IP Run-time	Optimality Gap
(20, 20)	0.744s	0.66 million	5.09s	**156**	**1.99s**	1.99s	0.0032s	0.0067%
(20, 30)	1.114s	1.05 million	7.29s	**182**	**4.15s**	4.15s	0.0015s	0.0010%
(20, 40)	1.248s	1.21 million	4.03s	**193**	**2.44s**	2.43s	0.0036s	0.0107%
(20, 50)	1.052s	1.00 million	2.25s	**185**	**1.67s**	1.67s	0.0039s	0.0068%
(20, 60)	1.643s	1.59 million	6.32s	**201**	**4.88s**	4.88s	0.0017s	0.0000%
(30, 20)	5.347s	4.16 million	26.96s	**326**	**14.90s**	14.90s	0.0029s	0.0052%
(40, 20)	35.084s	18.97 million	125.16s	**552**	**85.73s**	85.72s	0.0174s	0.0096%

Table 3.4: Run-time and Cost Comparison on the Framework in Different Modes with Changing Number of Server Types.

Problem Sizes	$GI/GI/n - IP$			$GI/GI/n$ – Reduced IP		
Number of Types	Config Time	Config Size	Run-time	Reduced Config Size	Run-time	Optimality Gap
30	3.089s	2.3 millions	51.90s	**359**	**16.93s**	0.0078%
40	4.060s	3.0 millions	30.43s	**316**	**13.05s**	0.0057%
50	5.249s	4.1 millions	308.86s	**421**	**120.11s**	0.0000%
60	6.008s	4.7 millions	308.75s	**402**	**103.99s**	0.0000%

cases. However, the run-time of the combinatorial models in our experiments are not close to the time limit of the solver. The bottleneck that limits the size of problem instances is the size of the configurations. Developing heuristics to reduce the size of the configuration set is an interesting direction of future works.

3.4.4 Full Framework Experiment

In order to test the entire framework, we generated a set of 36 job classes. Let the jobs in each class i have an expected inter-arrival time of $\frac{1}{\lambda_i}$ seconds with a standard deviation of $\frac{sd_i}{\mu_i}$ seconds. The expected lifetime of the jobs in class i is $\frac{1}{\mu_i}$ with a standard deviation of $\frac{sd_i}{\mu_i}$. The SLA of class i restricts the probability that a class i job waits for more than 10 seconds to less than or equal to α_i. The $\left(\frac{1}{\lambda_i}, \frac{1}{\mu_i}, sd_i, \alpha_i\right)$ settings of these 36 job classes are the Cartesian product of $\{0.3, 0.9, 1.5\} \times \{70, 170, 270\} \times \{2, 4\} \times \left\{\frac{1}{80}, \frac{1}{20}\right\}$. We experimented with the framework in two modes: $GI/GI/n$ queueing models with IP optimization and $GI/GI/n$ queueing models with the reduced IP model. We decided to use the $GI/GI/n$ model to approximate the service slot requirements since it approximates a solution that satisfies the SLA with fewer assumptions on the job behaviors in cloud systems compared to the $M/M/n$ models. With different number of server types, the run-times of the framework and the optimality gaps of the server set solution from different mode of framework are shown in the Table 3.4.

In Table 3.4, the optimality gap is calculated with respect to the solution of $GI/GI/n - IP$ model, i.e. Optimality Gap $= \frac{\text{Reduced IP Solution} - \text{Original IP Solution}}{\text{Original IP Solution}}$. Sim-

Table 3.5: Upper Bound of the 95% Confidence Interval on the Probability of Jobs in Each Class Waiting for More Than 10 Seconds in the Simulation Using the Solution of $GI/GI/n$ − Reduced IP.

Mean Inter-arrival and Lifetime $\left(\frac{1}{\lambda_i}, \frac{1}{\mu_i}\right)$	Standard Deviation Factor and SLA Probability Limit (sd_i, α_i)			
	(2, 0.0125)	(2, 0.05)	(4, 0.0125)	(4, 0.05)
(0.3, 70)	0.00178	0.00432	0.00959	0.01930
(0.3, 170)	0.00211	0.00571	0.01485	0.02319
(0.3, 270)	0.00236	0.00566	0.01510	0.03122
(0.9, 70)	0.00193	0.00439	0.01004	0.02058
(0.9, 170)	0.00166	0.00549	0.01549	0.02644
(0.9, 270)	0.00248	0.00617	0.01566	0.03218
(1.5, 70)	0.00187	0.00445	0.00906	0.02122
(1.5, 170)	0.00304	0.00688	0.01410	0.03035
(1.5, 270)	0.00291	0.00684	0.01821	0.03243

ilarly to the experiments of the combinatorial models, the run-time of the framework is always shorter when using the reduced IP model, and the framework with the original IP model produces the solution with a better cost. However, in case of the framework experiment, the amount of the time saved by the reduced IP model is more meaningful compared to the cost saved by the original IP model, which is always less than 0.01% in the experiment.

Notice the $GI/GI/n$ model introduced in Equation (2.4) approximates the service slot requirements in a $GI/GI/n$ queue, and the reformulated IP model also approximates the optimal solution of the combinatorial problem. Thus, we want to observe the performance of the cloud system simulator using the result of the framework with two sources of approximation errors. We constructed a simulator with first in first out (FIFO) scheduling strategy that has the server capacity and configuration calculated by the LP model with rounding. Then the simulator was run for 1,000 trials with a length of 100,000 seconds in each trial. The inter-arrival times and lifetimes of the jobs in the simulation are generated from lognormal distributions with the predefined mean and standard deviation. The upper bound of the 95% confidence interval on the expected probability of the jobs in each class waited for more than 10 seconds is shown in Table 3.5. Note the upper bound of the confidence interval exceeds the SLA restriction in the job classes with an expected lifetime ≥ 170 and high standard deviation. According to our observation, the simulation is more likely to generate jobs with long lifetimes in these classes, then the large jobs block all service slots and cause the waiting queue grow substantially. In such case, a large number of jobs wait in the queue, breaking the SLA. Using the shortest job first (SJF) scheduling strategy in the simulation will result in a lower average waiting time compared to

FIFO. Specifically, SJF scheduling strategy gives the highest priority to the jobs with the shortest lifetime, so the jobs with long lifetimes may experience extremely long waiting time. In our problem, the cloud want to complete every job. Thus, we choose to use FIFO in our simulator, since it is a fair scheduling strategy, which means it guarantees that all jobs will be scheduled eventually. The trade-off between the fairness and efficiency in cloud requires more study and is out of the scope of this work. The low waiting time on the jobs not in class k suggests an overestimation on the server capacity in our framework. The overestimation comes from the approximation from the $GI/GI/n$ model as discussed in the experiment of queueing models, and the reduced IP model guarantees of having enough service slots in its solution. Therefore, both approximation errors tend to overestimate the server capacity required in the cloud system, and the final solution of the framework satisfies the workload except for one class of jobs.

3.5 Conclusion

In this chapter, we introduced a framework to calculate the set of servers that guarantees SLAs in the cloud system with minimum operating cost, where the jobs are submitted and served by the cloud following some random process.

Specifically, we applied $M/M/n$ queues and $GI/GI/n$ queues as the queueing models in the framework, and an IP model and its linear relaxation with some rounding strategies are constructed to solve the packing problem in the second stage. Based on our experimental results, the framework using $GI/GI/n$ queueing models and an LP optimization model seems to be the most applicable setting in this work. It could produce a set of servers that satisfies the SLAs of a cloud system while assuming the inter-arrival times and lifetimes of the jobs in each class are i.i.d. and following some arbitrary distribution. The framework finds a solution in a short time with a cost that is almost the minimum.

According to the structure of this framework, the queueing model in stage I can be replaced by another type of queueing model, as long as the new model calculates a minimum service slot requirement that satisfies the SLAs. Similarly, the IP model in stage II can also be replaced by another solver for the two-dimensional packing problem. Therefore, this framework has a potential to be applied to more types of cloud systems by redefining the queueing models and the combinatorial model in the framework.

In this work, we have studied our framework with two queueing models that both assume single job arrivals and an infinite queue length. In future work, we pro-

pose to improved this framework by using more complex queueing models, such as queues with batch arrivals ($M^X/G/n$ queues [73]) and queues with finite queue length ($GI/GI/n/C$ queues [52]). In the combinatorial models, the size of the configurations is limiting the model from larger problem instances. The complexity of the models can be reduced with some machine learning algorithms or heuristics that compare the configurations for different server types and remove the non-promising ones without solving the linear relaxation.

Chapter 4

Short-term Case Study: Google Cloud with Borg System

4.1 Introduction

In this chapter, we test the robustness our short-term capacity planning framework with a public dataset from the Borg cluster management system in Google Cloud [59]. The full dataset contains the information of all events in multiple Google clusters on May 2 2019. The size of the original dataset is around 2.4TB, which is too large to analyze. The goal of our experiments is to observe the behavior of our framework with the workload in a real cloud, where the job arrivals and lifetimes have different patterns than the generated distributions we used in Chapter 3. Thus, we extracted the information of all events in one cluster on May 2 2019, resulting in a dataset including around 1.2 million tasks with a size of 8GB.

The jobs in Google Cloud [63] have a different definition than the jobs in our framework, so we call the jobs in Google Cloud *collections* to avoid ambiguity. In Google Cloud, each collection consists of multiple *tasks* that can be assigned to different machines. Collections may have constraints that force their tasks to be assigned to machines with particular attributes, such as processor architecture, OS version, or external IP address. However, the dataset does not contain the machine attribute information, so we ignore such constraints in this chapter. Similar to the jobs in our framework, each task has an independent CPU requirement, memory requirement, and lifetime in the system.

The Borg dataset contains the events of every task that arrived to the Borg system. Each event is described with its event type, timestamp, task id, collection id, CPU requirement, memory requirement, machine id, scheduling class, and priority. Table 4.1 shows all event types and their meaning, and Figure 4.1 (Figure 2 in the Google

Borg paper [63]) depicts the effect of the event on the task state. In all event types described in Table 4.1, only fail and lost denote the occurrence of system errors that cause the task to exit the system abnormally. Thus, we discard the tasks that are failed or lost, and Figure 4.2 shows the life cycle of the remaining tasks that are normally terminated. Each event has a timestamp in microseconds since 600 seconds before 00:00 May 1 2019, so an event that happened at 00:00:20 May 1 2019 has a timestamp of 620×10^6. The record of each event of a task contains the id of the task as the "task id", and "collection id" refers to the id of the collection that contains the task. The same task ids can be used in different collections to refer to different tasks, so we merge the "task id" and the "collection id" to create an unique id for each task, called the *uid*. Events with the same uid are grouped together to form the trace of each task. The CPU and memory requirements represent the amount of resource of the machine "machine id" that is occupied by the task until the next event of this task happens. The scheduling class is an integer number from 0 to 3 roughly representing the latency sensitivity of the task, where class 3 represents the most latency-sensitive task and class 0 represents the least latency-sensitive class. The priority is an integer number with the larger number meaning that the task has a higher priority considered by the scheduler.

With the dataset described above, we process the trace of each task to construct a job that fits our framework. Recall that we assume each job has an arrival time, lifetime, waiting time, CPU requirement, and memory requirement (see Section 3.2). Consider a task with a life cycle shown in Figure 4.2. The arrival time of the job is set to be the time of the first submit event of the task. The lifetime of the job is the total time that the task spent in state "Running", which is the sum of the time differences between every schedule event and its next evict, finish, or kill event. Similarly, the job waiting time is the total time that the task spent in state "Pending", where the submit, accept, and evict events enter the "Pending" state and the schedule and kill events exit the "Pending" state. The CPU and memory requirement of a job is assumed to be constant, but the CPU and memory requirement may be updated in the life cycle of the task. Thus, we round the CPU and memory requirement of the job to the maximum CPU and memory requirement in the life cycle of the task. The scheduling class and priority number are kept the same in the job, which are used later to classify the job. After these steps, we construct around 1.14 millions jobs for our experiments on the framework.

Table 4.1: Event Types with Meanings Explained in the Borg Dataset Repository [68].

SUBMIT	A task was submitted to the cluster manager.
QUEUE	A task is queued (deferred) until the scheduler is ready to act on it.
ENABLE	A task became eligible for scheduling; the scheduler will try to place the task as soon as it can.
SCHEDULE	A task was scheduled on a machine.
EVICT	A task was descheduled because of a higher priority task, because the scheduler overcommitted and the actual demand exceeded the machine capacity, because the machine on which it was scheduled became unusable (e.g., taken offline for repairs), or because the task's data was lost for some reason (this is very rare).
FAIL	A task was descheduled (or, in rare cases, ceased to be eligible for scheduling while it was pending) due to a user program failure of some kind such as a segfault, or a process using more memory than it requested.
FINISH	A task completed normally.
KILL	A task was cancelled by the user or a driver program, or the task's parent exited, or another task on which this job was dependent ended.
LOST	A task was presumably ended, but a record indicating its termination was missing from our source data. The event captures the moment when this was realized.
UPDATE_PENDING	A task's scheduling class or resource requirements were updated while it was waiting to be scheduled.
UPDATE_RUNNING	A task's scheduling class or resource requirements were updated while it was running (scheduled).

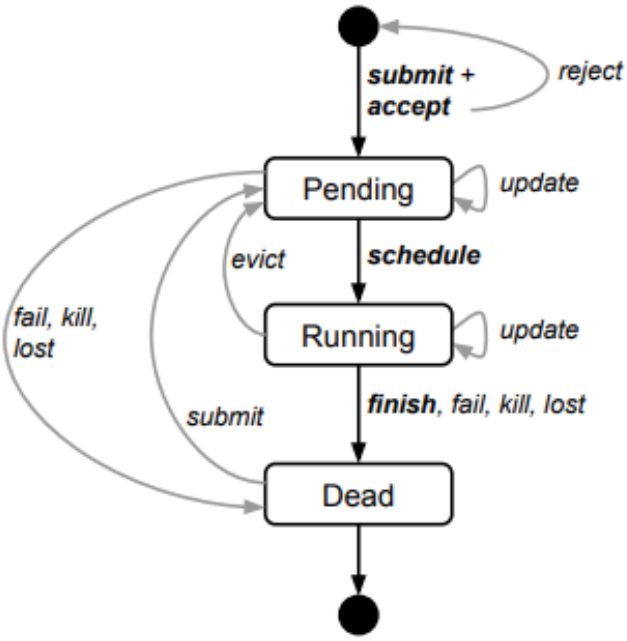


Figure 4.1: The state diagram for tasks in Google Cloud with Borg system (Figure 2 in the Google Borg paper [63])

.

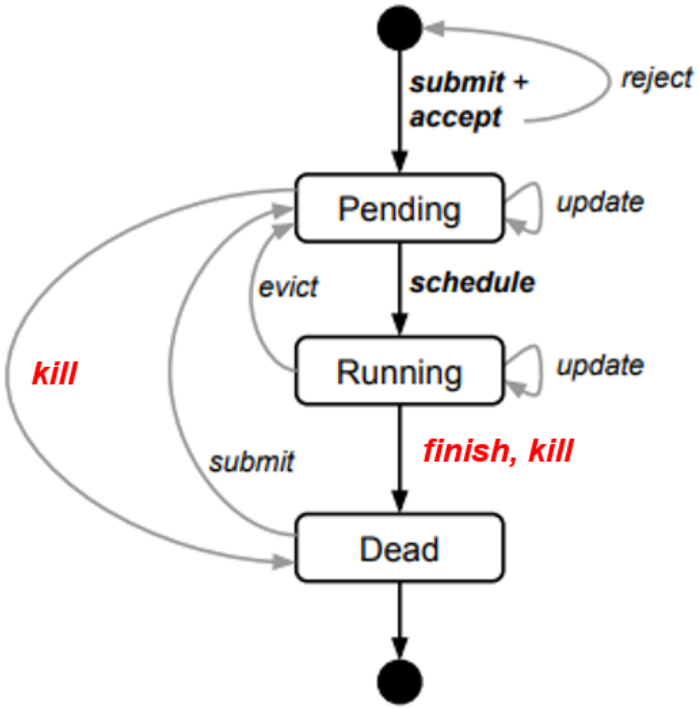


Figure 4.2: The state diagram for the normally terminated tasks.

4.2 Experiment Settings

4.2.1 Server Types

In our short-term server capacity planning framework, we assume the servers are grouped into different types based on their resource capacities and operating costs. The Google Cloud dataset does not provide the hardware settings or the costs of the servers due to security reasons. The history of the processed job in each machine is given, and the resource requirement of each job is denoted as the percentage of the CPU and memory it occupies in the assigned machine. Therefore, we assume all the servers have the same type, and their resource capacity is normalized as 1 CPU and 1 memory. We set the unit operating cost of the servers to be 1 as the cost is irrelevant when there is only one type of servers.

4.2.2 Job Classes

The jobs are assumed to be classified into different classes based on their service level agreements (SLAs) and resource requirements in the short-term server capacity planning framework. These parameters are not directly given in the dataset, so we pre-process the dataset for the parameters of each job class.

Service Level Agreement

The SLA of the jobs in a class states that the probability that a job waits for more than t seconds must be less than or equal to α (see Equation 3.1); it is a restriction on the job latency. The SLAs of the jobs are not given directly in the dataset due to security reasons. Thus, we assume the Borg system has achieved the hidden SLA, and we reproduce the SLA based on the observation of the job waiting times. The latency sensitivity of a job is given by its scheduling class and priority values. We define a set K of 12 priority levels for the jobs, from the Cartesian product of 4 scheduling classes and 3 ranges of priority values. We group the jobs with the same priority level $k \in K$, then we calculate the 90th percentile w_k of the waiting times of the jobs. Finally, we define the SLA of the jobs with priority level $k \in K$ as the probability of waiting for more than w_k seconds is less than or equal to 0.1 ($t = w_k, \alpha = 0.1$). The priority level is not equivalent to class, since the jobs in the same class should also have the same resource requirements.

Resource Requirements

We notice most of the jobs in the dataset have different resource requirements, and there are too many classes if each job belongs to a different class. Thus, we apply the k-means algorithm in the Python module scikit-learn [45] to cluster the jobs in each priority level based on their resource requirements. We run two experiments in this chapter: a 24-class experiment and a 34-class experiment. In the 34-class experiment, the k-means algorithm is set to produce 3 clusters of jobs in each priority level and results in 36 clusters of jobs in 12 priority levels. We discard the clusters with less than or equal to 20 jobs and have 34 clusters of jobs remaining. We define a job class for each cluster of jobs, where the job class has the resource requirement equal to the maximum resource requirement in the cluster. For example, if there are only two jobs in a cluster, one requires 3 CPU cores and 5 GB memory, and the other requires 5 CPU cores and 3 GB memory. Then the corresponding jobs class has a resource requirement of 5 CPU cores and 5 GB memory. The service slots for the jobs belonging to this class have the same size as the resource requirement. The 24-class experiment has the similar procedure for defining the job class, but sets the k-means algorithm to produce 2 clusters of jobs in each priority level, which results in 24 clusters of jobs in total. All of these clusters have more than 20 jobs, so the experiment has 24 classes of jobs.

Inter-arrival Time Distribution and Lifetime Distribution

In Chapter 3, we presented two queueing models for our framework: $M/M/n$ and $G/G/n$. We want to know which one fits better on the Borg dataset. The $M/M/n$ queue assumes the job inter-arrival time follows an exponential distribution and the job lifetime follows another exponential distribution (see Section 2.1.1). In Figure 4.3, we depict the inter-arrival time distribution and lifetime distribution of the jobs in two representative classes of the 24-class experiment. Figure 4.3a shows that most of the jobs in the class 1 of 24-class experiment have small inter-arrival time, which means a large proportion the jobs arrive to the system in batches. The lifetime of the jobs in this class is shown in Figure 4.3b, where the distribution has multiple peaks and a few of the jobs have a much longer lifetime than the others. The inter-arrival time distribution of the jobs in the class 2 of the 24-class experiment also has a peak around 0 inter-arrival time (see Figure 4.3c), which indicates the existence of batch arrivals. However, there are more jobs with larger inter-arrival time compared to the jobs in the class 1, so the size of the job arrival batches is smaller in the class 2 compared to the class 1. Most of the jobs in the class 2 have a similar short lifetime

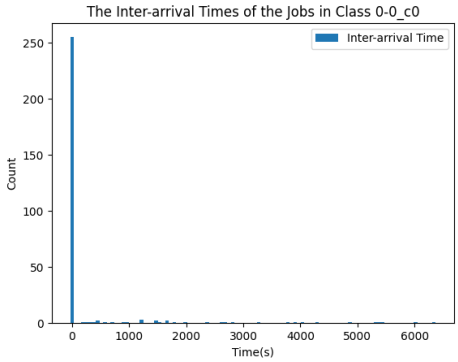
(see Figure 4.3d).

Figure 4.4 shows the inter-arrival time distribution and lifetime distribution of the jobs in two representative classes of the 34-class experiment. The jobs in class 1 of the 34-class experiment arrive to the cloud in batches, so most of them have small inter-arrival times (see Figure 4.4a). Figure 4.4b shows most of the jobs in the class 1 have a lifetime of less than 1000 seconds or around 2500 seconds. The distribution of the inter-arrival time and lifetime of the class 2 jobs in the 34-class experiment are shown in Figure 4.4c and Figure 4.4d respectively. The distribution of the job inter-arrival time is still more right skewed compared to the exponential distribution with the same mean. The lifetime distribution indicates most jobs have short lifetimes but with a few exceptions with much longer lifetimes. None of the classes in the 24-class experiment or 34-class experiment have both their job inter-arrival time and job lifetime exponentially distributed, which violates the assumption of an $M/M/n$ queue. Therefore, we use $G/G/n$ queueing models in both experiments, since $G/G/n$ queue assumes the job inter-arrival time and lifetime are arbitrarily distributed (see Section 2.1.1).

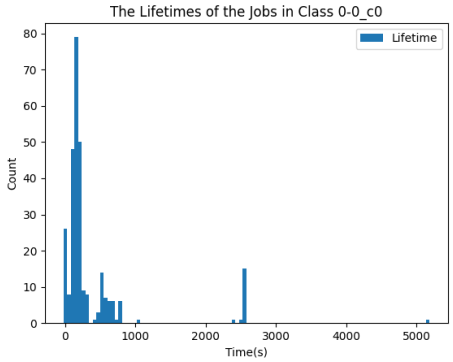
4.3 Experimental Results

Two experiments with different numbers of classes are run in this section: the 24-class and 34-class experiment. Each experiment has a calculation part and a simulation part. In the calculation part, the server capacity planning framework (see Section 3.3) is used to optimize the server requirement and the configuration of each server. The framework uses the $G/G/n$ queueing model to calculate the number of service slots required by each class. Then it generates the set of non-dominated server configurations (see Section 3.3.3). The framework uses the integer programming (IP) model (3.2)-(3.6) or the linear programming (LP) model (3.7)-(3.11) with rounding to calculate the server requirement and the server configuration used in the simulations. The overview of calculation results from different optimization models in both experiments is shown in Table 4.2. We compare the optimal solution and complexity to analyze the pros and cons of the optimization models and the classification strategies used in the experiments.

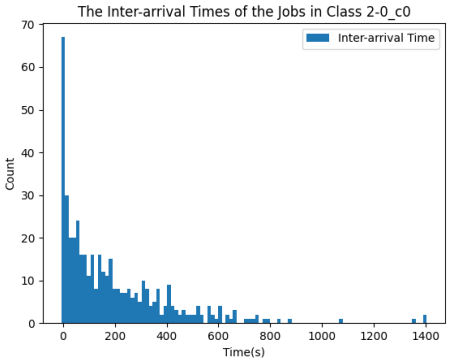
In Table 4.2, the "Config Time" corresponds to the time spent on generating the set of non-dominated configurations, and the "Config Size" corresponds to the cardinality of the configuration set, that is the number of non-dominated configurations ($\sum_{j\in J} |\bar{B}_j|$). The 24-class experiment generates around 0.33 million configurations in 0.31 seconds, and the 34-class experiment generates around 45.86 millions configura-



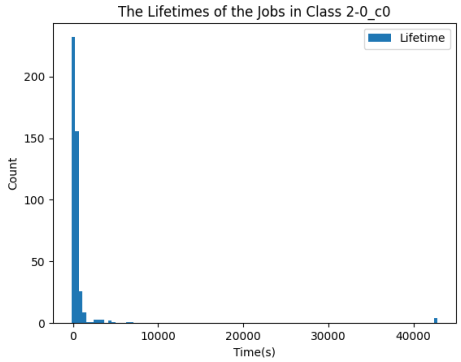
(a) The distribution of the class 1 job inter-arrival times.



(b) The distribution of the class 1 job lifetimes.

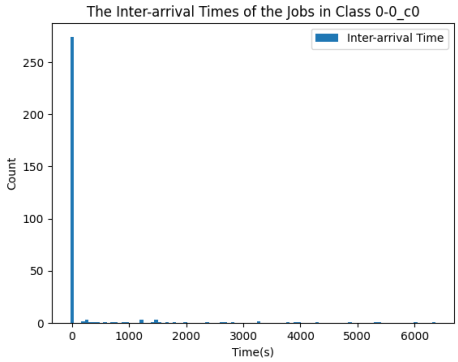


(c) The distribution of the class 2 job inter-arrival times.

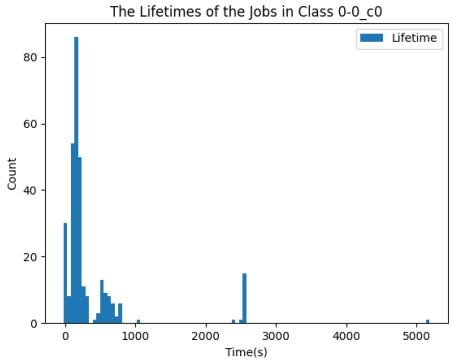


(d) The distribution of the class 2 job lifetimes.

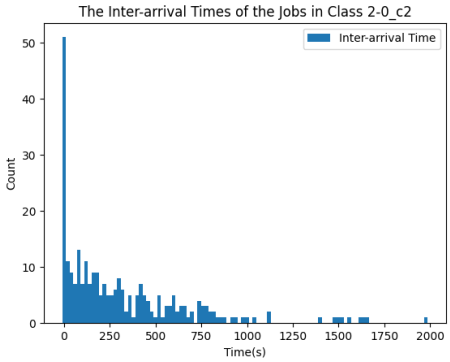
Figure 4.3: The inter-arrival time and lifetime distributions of the jobs in different classes in the 24-class experiment.



(a) The distribution of the class 1 job inter-arrival times.



(b) The distribution of the class 1 job lifetimes.



(c) The distribution of the class 2 job inter-arrival times.

(d) The distribution of the class 2 job lifetimes.

Figure 4.4: The inter-arrival time and lifetime distributions of the jobs in different classes in the 34-class experiment.

tions in 60.13 seconds. The set of configurations grows exponentially as the number of classes increases, and the experiment with 4 classes of jobs in each priority level exceeds our memory limit. The "Runtime" row shows the runtime of each optimization model, where the runtime shown for the LP model is the sum of the LP runtime and the runtime of the small IP model for rounding. As expected, the LP model with rounding is always faster than the IP model in both experiments.

Both optimization models calculate the same number of servers required in the cloud system shown in the "Server Requirement" row. It is a single number since there is only one type of server in each experiment. The 34-class experiment with the jobs classified in more detail produces a solution that saves a significant number of servers required compared to the solution of the 24-class experiment. The difference in their solutions is caused by the error introduced when we classify the jobs and round their resource requirements to the maximum resource requirement in each class. With more classes, each class in the 34-class experiment is smaller, and the gap from the job resource requirement to the maximum in the class is smaller compared to the 24-class experiment. Thus, the resource requirements of the jobs in the 34-class experiment are less overestimated and leads to a smaller server requirement compared to the 24-class experiment.

We evaluate the solution of each experiment with a simulator that mimics the Google Cloud system to check the effectiveness of our framework. In the cloud system simulation, we use the number of servers and their configurations of service slots calculated by the server planning framework. The jobs arrive to the simulated system in the same pattern as the jobs arrive to the real cloud in May 2 2019. For example, if a job entered the Google Borg system at 00:00:10 May 2 2019, then the simulation would experience the same job entering at 10 seconds after the start with the same lifetime and resource requirement. After each job arrival, the simulation makes the scheduling decision based on the scheme shown in Figure 4.5. When a class $i \in I$ job arrives to the system, if there is an idling class i service slot in a server, then the job is assigned to this server, otherwise the job is appended to the end of the waiting queue for the class i jobs. The jobs in the class i waiting queue are ranked by their arrival time (FIFO), and whenever a class i job terminates, the oldest job in the waiting queue is assigned to that slot. The job waiting time is recorded in the simulation when a job leaves a waiting queue. We summarize the job waiting times in the simulation to check the satisfaction of the SLAs by our framework solutions. As explained in Section 4.2.2, we define the SLA of the jobs in each class $i \in I$ as: the probability of waiting more than w_i is less than or equal to 0.1, where w_i is the 90th percentile of the true job waiting time from the dataset in this class. The

Table 4.2: Numerical Results of the 24-class and 34-class Experiments

	24-class Experiment		34-class Experiment	
Config Time	0.31 second		60.13 seconds	
Config Size	0.33 million configurations		45.86 million configurations	
Optimization Model	IP Model	LP Model	IP Model	LP Model
Runtime	1.91 seconds	1.71 seconds	65.25 seconds	40.87 seconds
Server Requirement	32493 servers	32493 servers	19138 servers	19138 servers

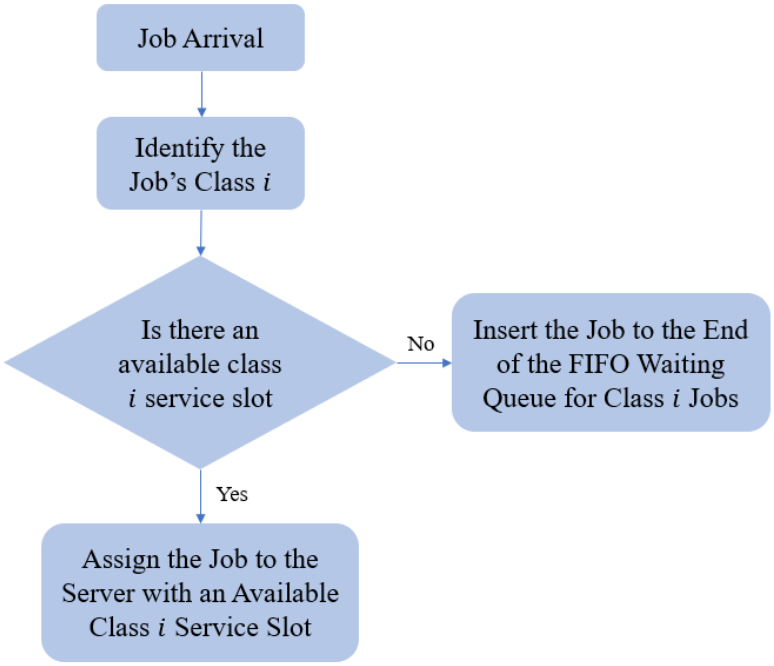


Figure 4.5: The scheme of the cloud system simulation scheduling strategy.

probability of jobs waiting more than the time threshold in the simulator with the 24-class experiment solution is presented in Table 4.3. The job waiting time summary of the 34-class experiment simulation is presented in Table 4.4.

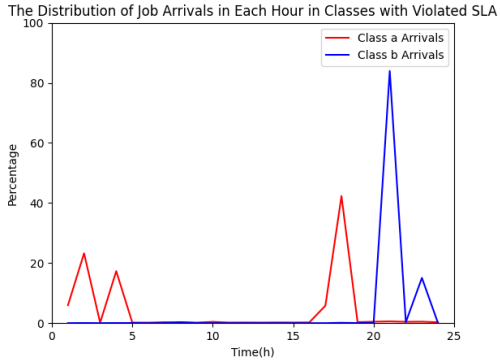
In Table 4.3, the 12 priority levels are listed as rows, where the jobs with the larger priority level have the higher priority. In the 24-class experiment, the jobs in each priority is grouped into two classes: Class 0 and Class 1. The number of jobs in each class is shown in the "Job Count" column, and the "PW" column contains the probability of a job waiting more than the time threshold in each class in the simulation. The simulation results that violate the SLAs are colored red. We present the workload of two job classes with violated SLAs in 24-class experiment simulation in Figure 4.6a. The class a and the class b in Figure 4.6a correspond to the class 1 in priority level 2 and the class 1 in priority level 5 respectively. We observe high peaks in the workload of these classes. Moreover, the class 1 in priority level 5 (class

Table 4.3: Probability of the jobs waiting more than the time threshold in each class in the 24-class experiment. The probability should be less than or equal to 0.1 to satisfy the SLA (PW$\leq$ 0.1).

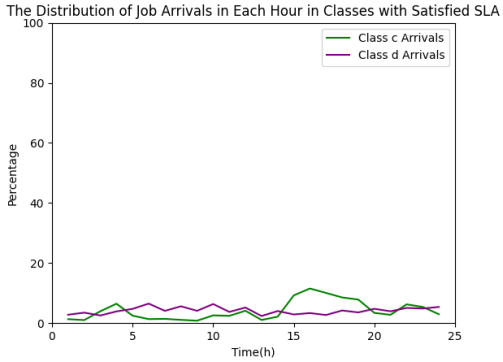
Priority Level	Class 0		Class 1	
	Job Count	PW	Job Count	PW
1	291	0.7216	1781	0.8304
2	493448	0.5549	28544	0.2029
3	25031	0.0000	89	0.0000
4	26848	0.7780	6561	0.7683
5	201623	0.0000	2842	0.9018
6	38171	0.0000	253040	0.0000
7	443	0.0000	68	0.0000
8	3684	0.0000	182	0.0330
9	593	0.0000	8964	0.0000
10	17694	0.4855	2878	0.0653
11	400	0.0000	1452	0.0000
12	2867	0.0000	27110	0.0000

b) has the SLA violated the most (see Table 4.3), and its workload has a higher peak compared to the workload of the class 1 in priority level 2 (class a). In contrast, the workload of the class 0 in priority level 3 (class c) and the class 0 in priority level 6 (class d) that have their SLAs satisfied in the simulation is shown in Figure 4.6b, where the job arrivals are more even in the day compared to the job classes with violated SLAs. Thus, we conclude the peak workload overloads the service slots allocated for the class and cause the jobs to wait in the queue, since our framework does not share the service slots with different classes.

Table 4.4 shows the simulation results of the 34-class experiment. The experiment has three classes of jobs in each of the 12 priority levels. The two classes with less than 20 jobs are discarded and denoted as "N/A" in the table. The job classes with their SLAs violated in the simulation are colored red, and the workload of two representative classes is presented in Figure 4.7a. The simulation of the 34-class experiment shows similar results compared to the 24-class experiment. Figure 4.7a shows the classes with violated SLAs have dense workload in few short periods, where the class a and class b in the figure are the class 1 in priority level 5 and the class 2 in priority level 8 respectively. The class 2 in priority level 8 (class b) is the class that has the shortest waiting time with violated SLAs (see Table 4.4), and it has the workload with multiple peaks that is sparser than the workload of the other class with violated SLA. Figure 4.7b shows the classes with satisfied SLAs have even workload among longer periods, where the class c and class d in the figure are the class 0 in priority level 3 and the class 0 in priority level 6 respectively. Therefore, the results of the 34-class experiment support our previous conclusion: the peak workload overloads



(a) The per hour workload of two classes that have violated the SLAs in the simulation of the 24-class experiment.

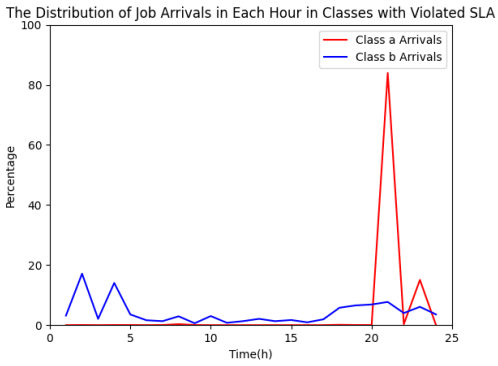


(b) The per hour workload of two classes that have satisfied the SLAs in the simulation of the 24-class experiment.

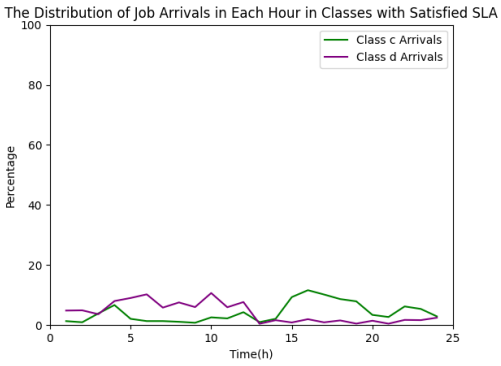
Figure 4.6: The per hour workload of the representative job classes in the 24-class experiment.

Table 4.4: Probability of the jobs waiting more than the time threshold in each class in the 34-class experiment. The probability should be less than or equal to 0.1 to satisfy the SLA (PW$\leq$ 0.1).

Priority Level	Class 0		Class 1		Class 2	
	Job Count	PW	Job Count	PW	Job Count	PW
1	314	0.7389	1755	0.8279	3	N/A
2	404587	0.5833	27971	0.2053	89434	0.7346
3	20704	0.0000	85	0.0000	4331	0.0000
4	25460	0.7689	4446	0.7501	2903	0.8219
5	169224	0.0000	2842	0.9018	32399	0.0000
6	251524	0.0000	38902	0.0000	785	0.0000
7	54	0.0000	187	0.0000	270	0.0000
8	2412	0.0000	157	0.0382	1297	0.1180
9	1523	0.0000	287	0.0000	7747	0.0000
10	17629	0.4852	2876	0.0650	67	0.0000
11	1419	0.0000	424	0.0000	9	N/A
12	26081	0.0000	3279	0.0000	617	0.0000



(a) The per hour workload of two classes that have violated the SLAs in the simulation of the 34-class experiment.



(b) The per hour workload of two classes that have satisfied the SLAs in the simulation of the 34-class experiment.

Figure 4.7: The per hour workload of the representative job classes in the 34-class experiment.

the system in a short period and causes a large number of jobs waiting in the queue.

4.4 Conclusion

Our analysis of the Google Cloud Borg dataset shows that some of the assumptions in our framework are too strong for the real cloud systems. The framework assumes the jobs in the same class have the same resource requirements. In practice, many jobs may have similar resource requirements but not the same. If the jobs with different resource requirements belong to different classes, then the number of job classes is too large for the framework. In the experiments, we cluster the jobs with similar resource requirements into a class. Then we set the size of the service slots to be the largest job size in each class, so the service slot can process every job in the class. In Table 4.3 and Table 4.4, we observe most of the job classes in the 24-class experiment and the 34-class experiment have 0 probability of waiting more than the time threshold (PW$=0$), but the SLA allows the probability to be less than or equal to 0.1 (PW≤ 0.1). Hence, the solution of our framework overestimates the server requirements. In Table 4.2, the total server requirement calculated in the 34-class experiment is smaller than the total server requirement in the 24-class experiment. As we increase the number of clusters in the k-means algorithm that classify the jobs into different classes, the jobs in each cluster have closer resource requirement and the rounding error is reduced, so the server requirement overestimation is also reduced. Thus, we can increase the number of job classes to improve the accuracy of the server requirement calculation.

Our framework makes another strong assumption that the workload is stable over

time in the cloud. In the dataset, the workload is imbalanced throughout the day (see Figure 4.6a and 4.7a). In such cases, the solution of the framework cannot guarantee the SLAs in each class, especially when the job arrivals are too dense. A possible improvement is to allow some resource sharing. For example, if most of the service slots for a class are idling, then a proportion of these service slots can be used for the jobs in other classes. Another strategy is to apply the framework on a shorter period, such as hours, and solve the framework more frequently. However, the workload in each hour may still be unstable, so a real-time scheduler based on heuristics or machine learning algorithms can potentially improve the solution of our framework.

For further improvements, we observe the majority of the job inter-arrival times are close to 0 from Figure 4.3 and Figure 4.4, suggesting that the jobs tend to arrive in batches. Adapting the queueing model in the framework to consider batch arrivals (i.e. $GI^X/GI/n$ queue, where X is the random variable representing the size of batches) will improve the accuracy of our framework on real cloud server management problems. From the experiment with real cloud data, we observe the deterministic parameters, including the job resource requirements, assumed by the framework are hard to satisfy in practice. This mismatch cause the framework to overestimate the server requirements in order to guarantee the SLAs. A potential improvement is to consider the problem in the second stage of the framework as a Stochastic Bin Packing Problem [13], which allows the size of the items to be represented by random variables.

Chapter 5

Long-term Purchase Plan Optimization in Data Centers

5.1 Introduction

In Chapter 3, we introduced a framework that manages server capacities and allocation strategies for cloud systems in the short-term. However, most cloud systems are intended to operate for a long time horizon: months, years, or decades. In a long-term server planning problem, the workload on the cloud system may change over time and the servers in the cloud system may break down. Thus, a server purchase plan is required in each period to maintain an adequate server capacity that will meet the service level agreements in the cloud system. In addition, the servers are non-homogeneous, having different performance for different types of arriving workloads. For example, servers with more CPU cores perform better on jobs with heavy computational loads and small memory requirements, while servers with larger memory size are more efficient on executing jobs with large memory requirements and small computational loads. Therefore, the set of servers that has the best performance given the workload in one period may have a disadvantage for future workloads. Thus, finding the best server purchasing plan that minimizes the long-term cost in the cloud is an interesting and challenging problem.

In past studies, server planning in cloud systems with changing workload [39, 44, 50] has focused on calculating the optimal server capacity plan that minimizes the power consumption or the operating cost, without considering the cost of purchasing servers. In Roy, Dubey, and Gokhale's work [49], cloud systems are assumed to lease machines instead of purchasing them. Thus, the decisions made in each period will not affect the cloud system in the future.

Long-term workforce and inventory planning problems were studied in control the-

ory for service and manufacturing industries [21, 25, 27, 40]. However, these traditional industries have different constraints compared to planning problems considered in the cloud. In manufacturing industries, inventories are used to cover the demands in each period [27, 40], while the number of servers in cloud systems does not decrease as time passes unless they break down. In the service industries, many applications focus on optimizing recruitment [21, 25], where each employee handles one task at a time. Servers in cloud systems are able to process multiple jobs simultaneously depending on their resource capacities. This assumption complicates the server demand calculation as studied in Chapter 3. In addition, employees in service industries may learn new skills or be promoted to provide services to more kinds of customers. This phenomenon is studied as learning behavior [21]. Based on the information from our cloud cooperator, the performance of servers in the cloud depends on their hardware configurations, and the hardware configurations do not change over time. Hence, the learning behavior does not apply to cloud systems.

In this chapter, we first define the long-term server purchase plan optimization problem based on the assumptions relevant in cloud systems. Then, we introduce a linear programming (LP) model that calculates the optimal solution to the problem and propose a myopic approximation for the LP model to reduce its computational complexity. We prove the myopic approximation is optimal when the workload is non-decreasing over the time horizon. To show the usefulness of our approach, we conducted experiments that demonstrate the advantage of the myopic approximation in terms of computational complexity and show the accuracy of the myopic approximations when the workload is decreasing in some periods.

5.2 Problem Definition

Cloud systems exhibit time changing workload that requires periodic adjustment of their capacity. Specifically, cloud operators update the system by purchasing a set of servers in each *period* to satisfy the workload. Thus, finding the optimal set of servers that would minimize the total expenditures over a multi-period planning horizon is an important problem. We refer to the problem of periodically adjusting capacity over the planning horizon as the long-term purchase plan optimization problem (LPP-OP). The structure of the cloud system is assumed to be the same as in the short-term setting (see Section 3.2). Servers are grouped into different *types* based on their resource capacities and costs. We denote with J the set of all server types. The jobs arriving to the system are differentiated into *classes* by their resource requirements, and we let I be the set of all job classes.

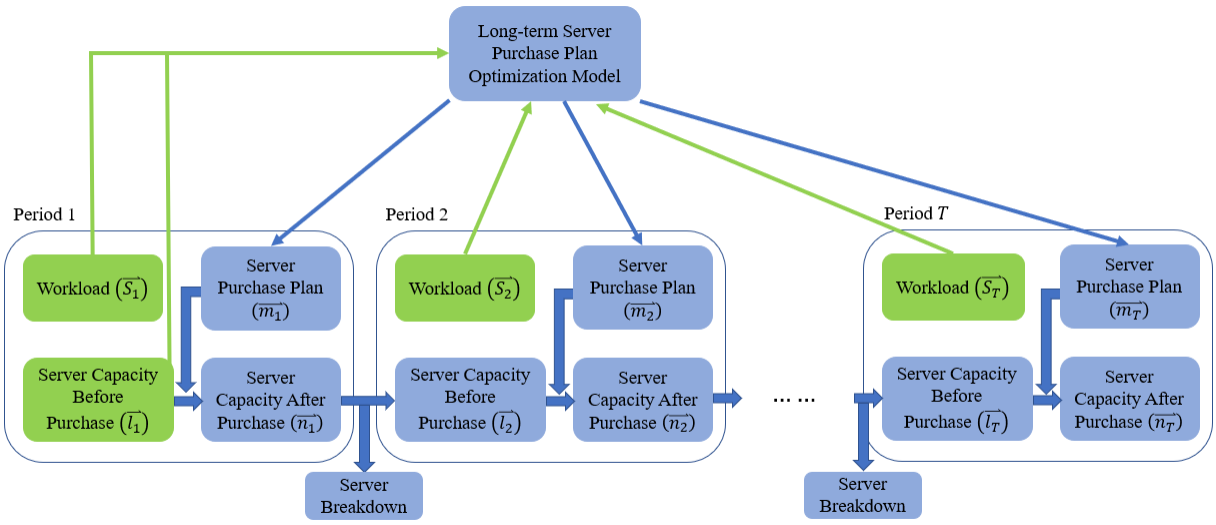


Figure 5.1: Scheme of Long-term Purchase Plan Optimization Problem

Figure 5.1 describes the scheme of the LPP-OP. We assume the planning horizon to be discrete, with $0 < T < \infty$ being the length of the time horizon. In each period, the cloud system experiences a specific workload, and the system may purchase a set of servers to update its server capacity to satisfy the workload. Over each period, some servers stop functioning and are summarized as the server breakdown in this period. A long-term server purchase plan optimization model calculates a server purchase plan for the cloud in each period that minimizes the total cost of the cloud, considering the workload in the planning horizon. Specifically, in period $t \in \{1, ..., T\}$, the cloud system has a set of servers $\vec{l_t} = (l_{1,t}, ..., l_{|J|,t})$ at the beginning of the period, and the system updates the number of servers to $\vec{n}_t = (n_{1,t}, ..., n_{|J|,t})$ after purchasing $\vec{m}_t = (m_{1,t}, ..., m_{|J|,t})$ servers. Thus, $\vec{n}_t = \vec{l_t} + \vec{m}_t$ in all periods t. We denote the overall server purchasing plan as a matrix $M = (\vec{m}_1, ..., \vec{m}_t)$, and the server capacity plan is defined as the matrix $N = (\vec{n}_1, ..., \vec{n}_m)$. At the end of each period, we assume a proportion of the type $j \in J$ servers break down, which is denoted by $0 \leq d_j \leq 1$, and therefore,

$$l_{j,t} = (1 - d_j)n_{j,t-1}, \forall j \in J, t \in \{2, ..., T\}.$$

The number of servers that the cloud system has at the beginning of the first period $\vec{l_1} = (l_{1,1}, ..., l_{|J|,1})$ is a parameter given in the problem definition.

In order to solve the LPP-OP, we use a dynamic optimization model to calculate the optimal server purchase plan that minimizes the total costs in the time horizon T. Specifically, we construct a total cost function that evaluates two time-dependent sources of costs in the cloud system: operating cost and purchasing cost.

Operating cost is the cost for operating servers that are owned by the cloud, including energy cost and maintenance cost. The operating cost in a period t is assumed to be calculated by a function $O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t)$, where $\vec{S}_t = (s_{1,t}, ..., s_{|I|,t})$ represents the workload in period t, and $s_{i,t}$ is the class $i \in I$ service slot requirement in period t. According to the experimental results in Chapter 3, the linear programming (LP) model (3.7)-(3.11) with rounding calculates a near-optimal solution (optimality gap $\leq 0.02\%$) in less time compared to the integer programming (IP) model (3.2)-(3.6). Thus, we define the operating cost function as the optimal objective value of the following LP model:

$$O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t) = \min \sum_{j \in J} o_j z_j \tag{5.1}$$

$$\text{s.t. } z_j \leq n_{j,t}, \qquad \forall j \in J \tag{5.2}$$

$$\sum_{j \in J} \sum_{\mathbf{v} \in \bar{B}_j} v_i x_{j\mathbf{v}} \geq s_{i,t}, \qquad \forall i \in I \tag{5.3}$$

$$\sum_{\mathbf{v} \in \bar{B}_j} x_{j\mathbf{v}} = z_j, \qquad \forall j \in J \tag{5.4}$$

$$z_j \in \mathbb{R}_+, \qquad \forall j \in J \tag{5.5}$$

$$x_{j\mathbf{v}} \in \mathbb{R}_+, \qquad \forall j \in J, \mathbf{v} \in \bar{B}_j \tag{5.6}$$

where

- $n_{j,t}$ is the number of type j servers owned by the cloud system after the purchasing in period t.
- $\vec{S}_t = (s_{1,t}, ..., s_{|I|,t})$ is the workload in period t, represented by the service slot requirement (see in Section 3.3.1) on each job class $i \in I$.
- o_j is the operating cost of each type j server in the system.
- $\bar{B}_j$ is the set of non-dominated configurations of type j servers (see Section 3.3.3).
- z_j is the number of type j servers operating in the system.
- $x_{j\mathbf{v}}$ is the number of type j servers that is operating as configuration $\mathbf{v}$.

The LP model above is the LP model in the short-term problem (Equation (3.7)-(3.11)) with one additional constraint, and we name this model as the operating cost model in this chapter. Constraints (5.3)-(5.6) are same as (3.8)-(3.11) in the short-term LP model. The new constraint (5.2) upper bounds the number of operating servers by the number of servers owned by the cloud system in each period.

Purchasing cost refers to the cost of purchasing the servers in each period. We use $h_{j,t}$ to denote the unit price of each type $j \in J$ server in period t, while $m_{j,t}$ denotes the number of type j servers to be purchased in the period. The purchasing cost of all servers in period t is $\sum_{j\in J} h_{j,t}m_{j,t}$. Table 5.1 summarizes the notation used when defining the total cost function. Thus, the total cost in period t can be formulated as

$$O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t) + \sum_{j\in J} h_{j,t}m_{j,t}.$$

For the total cost over the entire time horizon T, the cost in the future is usually less valuable compared to the current cost for the cloud operators due to the inflation rate and the investment interests. To take this into account, we use a discount factor $\beta \in (0, 1]$ in the total cost function

$$\sum_{t=1}^{T} \beta^{t-1} \left(O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t) + \sum_{j\in J} h_{j,t}m_{j,t} \right). \tag{5.7}$$

The future cost is weighted the same as the current cost when $\beta = 1$. Therefore, the optimal solution $\{m^*_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ of the LPP-OP can be obtained by solving the following dynamic programming (DP) model

$$\min \left[\sum_{t=1}^{T} \beta^{t-1} \left(O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t) + \sum_{j\in J} h_{j,t}m_{j,t} \right) \right] \tag{5.8}$$

$$\text{s.t. } l_{j,t} = (1 - d_j)n_{j,t-1}, \qquad \forall j \in J, t \in \{2, ..., T\} \tag{5.9}$$

$$n_{j,t} = l_{j,t} + m_{j,t}, \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.10}$$

$$m_{j,t} \in \mathbb{R}_+. \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.11}$$

The objective (5.8) of this DP model is to minimize the total cost in all periods. Constraints (5.9)-(5.10) enforce the rules of the server transitions defined in the scheme (Figure 5.1). Constraint (5.11) is the domain constraint that restricts the system to purchase a non-negative number of servers in each period.

To solve the DP model, we formulate an LP model and show that the two models are equivalent. Then we develop a myopic approximation for the problem with less computational complexity compared to the LP model. We prove the myopic approximation is equal to the optimal solution under additional assumptions.

Table 5.1: Notation

$T \in \mathbb{Z}_+$	The number of periods that the long-term capacity planning should cover.
$\beta \in (0, 1]$	The discount factor of the future costs.
I	The set of all job classes.
$s_{i,t} \in \mathbb{Z}_+$	The number of class i service slot required in period t, $i \in I, t \in \{1, ..., T\}$.
J	The set of all valid server types.
$h_{j,t} \in \mathbb{R}_+$	The unit price of a type j server in period t, $j \in J, t \in \{1, ..., T\}$.
$d_j \in [0, 1]$	The breakdown rate of servers in type j, $j \in J$.
$l_{j,t} \in \mathbb{R}_+$	The number of type j servers in the cloud system at the beginning of period t, $j \in J, t \in \{1, ..., T\}$.
$M \in \mathbb{R}_+^{\lvert J\rvert \times T}$	The server purchasing plan, where each element $m_{j,t}$ denotes the number of type j servers to purchase in period t, $j \in J, t \in \{1, ..., T\}$.
$N \in \mathbb{R}_+^{\lvert J\rvert \times T}$	The server capacity plan, where each element $n_{j,t}$ denotes the number of type j servers in the cloud system after the purchasing in period t, $j \in J, t \in \{1, ..., T\}$.

5.3 Linear Programming Model for Long-term Purchase Plan Optimization Problem

In this section, we transform the DP model (5.8)-(5.11) to a linear programming (LP) model, since the operating cost function $O(\cdot)$ is defined as the optimal objective value of an LP model, which makes the full DP model hard to solve. The new LP model includes the constraints of the operating cost functions $O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t)$ in different periods $t \in \{1, ..., T\}$. Moreover, we prove that the optimal solution of the LP model is also the optimal solution of the DP model (5.8)-(5.11). The new LP

model is defined as:

$$\min \sum_{t=1}^{T} \beta^{t-1} \left(\sum_{j \in J} o_j z_{j,t} + \sum_{j \in J} h_{j,t} m_{j,t} \right) \tag{5.12}$$

$$\text{s.t. } n_{j,t} = l_{j,t} + m_{j,t}, \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.13}$$

$$l_{j,t} = (1 - d_j) n_{j,t-1}, \qquad \forall j \in J, t \in \{2, ..., T\} \tag{5.14}$$

$$\sum_{j \in J} \sum_{\mathbf{v} \in \bar{B}_j} v_i x_{j\mathbf{v},t} \geq s_{i,t}, \qquad \forall i \in I, t \in \{1, ..., T\} \tag{5.15}$$

$$z_{j,t} \leq n_{j,t}, \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.16}$$

$$\sum_{\mathbf{v} \in \bar{B}_j} x_{j\mathbf{v},t} = z_{j,t}, \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.17}$$

$$z_{j,t} \in \mathbb{R}_+, \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.18}$$

$$x_{j\mathbf{v},t} \in \mathbb{R}_+, \qquad \forall j \in J, \mathbf{v} \in \bar{B}_j, t \in \{1, ..., T\} \tag{5.19}$$

$$m_{j,t} \in \mathbb{R}_+. \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.20}$$

The objective (5.12) of this LP model is to minimize the total cost in all periods. Constraints (5.13)-(5.14) are same as (5.9)-(5.10) in the DP model that enforce the server transition rules. Constraints (5.15)-(5.17) include constraints (5.2)-(5.4) in the operating cost model in all periods $t \in \{1, ..., T\}$. Thus, these constraints guarantee the service slot requirements are satisfied in all periods. Constraints (5.18)-(5.20) are the domain constraints.

To simplify the proofs, we first define the feasibility of the DP model (5.8)-(5.11).

Definition 5.1. *A solution $\{m_{j,t}, z_{j,t}\}_{j \in J, t \in \{1,...,T\}}$ is feasible for the DP model (5.8)-(5.11) if and only if:*

1. *The purchase plan is always non-negative, i.e.*

$$m_{j,t} \geq 0, \quad \forall j \in J, t \in \{1, ..., T\}$$

so the domain constraint (5.11) is satisfied.

2. *Suppose $N = \{n_{j,t}\}_{j \in J, t \in \{1,...,T\}}$ is the corresponding server capacity of the server purchase plan M that satisfies the constraints (5.9)-(5.10) in the DP model, i.e.*

$$n_{j,t} = l_{j,t} + m_{j,t}, \qquad \forall j \in J, t \in \{1, ..., T\}$$
$$l_{j,t} = (1 - d_j) n_{j,t-1}. \qquad \forall j \in J, t \in \{2, ..., T\}$$

Then $\{z_{j,t}\}_{j \in J}$ is a feasible solution in the operating cost model $O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t)$ in each period t.

Now we prove two propositions that are later used in our main result.

Proposition 1. *If a solution* $\{m_{j,t}, z_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ *is feasible for the DP model (5.8)-(5.11), then it is also feasible for the LP model (5.12)-(5.20).*

Proof of Proposition 1. Consider the feasible solution $\{m'_{j,t}, z'_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ of the DP model (5.8)-(5.11) in all periods, where the corresponding server capacities $\{n'_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ are restricted by the constraints (5.9)-(5.10) in the DP model

$$\begin{aligned} n'_{j,t} &= l_{j,t} + m'_{j,t}, && \forall j \in J, t \in \{1, ..., T\} \\ l_{j,t} &= (1 - d_j)n'_{j,t-1}. && \forall j \in J, t \in \{2, ..., T\} \end{aligned}$$

We know $\{z_{j,t}\}_{j\in J}$ is feasible in the operating cost model $O(n'_{1,t}, ..., n'_{|J|,t}; \vec{S}_t)$ in any period t by Definition 5.1.

Since the constraints (5.15)-(5.19) are the union of the constraints (5.2)-(5.6) in all periods t, the solution $\{z'_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ also satisfies the constraints (5.15)-(5.19). Notice the constraints (5.13)-(5.14) and (5.20) are the same as (5.9)-(5.11) in the DP model. Therefore, $\{m'_{j,t}, z'_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ is feasible in the LP model (5.12)-(5.20). ■

Proposition 2. *The optimal solution* $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ *of the LP model (5.12)-(5.20) is feasible for the DP model (5.8)-(5.11).*

Proof of Proposition 2. Consider the optimal solution $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ of the LP model (5.12)-(5.20) and its corresponding server capacities $\{n^*_{j,t}\}_{j\in J,t\in\{1,...,T\}}$. By constraints (5.13)-(5.14), we get that

$$\begin{aligned} n^*_{j,t} &= l_{j,t} + m^*_{j,t}, && \forall j \in J, t \in \{1, ..., T\} \\ l_{j,t} &= (1 - d_j)n^*_{j,t-1}. && \forall j \in J, t \in \{2, ..., T\} \end{aligned}$$

The domain constraint (5.20) in the LP model guarantees the optimal server purchase plan M^* is non-negative in all elements.

Now we want to show $\{z^*_{j,t}\}_{j\in J}$ is a feasible solution in $O(n^*_{1,t}, ..., n^*_{|J|,t}; \vec{S}_t)$ for all $t \in \{1, ..., T\}$. Notice the constraints (5.2)-(5.6) is a subset of the constraints (5.15)-(5.19), so $\{z^*_{j,t}\}_{j\in J}$ is also a feasible solution for $O(n^*_{1,t}, ..., n^*_{|J|,t}; \vec{S}_t)$.

Therefore, the optimal solution $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ of the LP model is feasible for the DP model (5.8)-(5.11). ■

Theorem 5.2. *Suppose* $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ *is the optimal solution of the LP model (5.12)-(5.20). Then* $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J,t\in\{1,...,T\}}$ *is also the optimal solution of the DP model (5.8)-(5.11).*

Proof of Theorem 5.2. Now consider the optimal solution $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ of the LP model (5.12)-(5.20). Let θ^* represent the optimal objective value of the LP model, i.e.

$$\theta^* = \sum_{t=1}^{T} \beta^{t-1} \left(\sum_{j\in J} o_j z^*_{j,t} + \sum_{j\in J} h_{j,t} m^*_{j,t} \right). \tag{5.21}$$

By Proposition 2, we know $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ must be feasible for the DP (5.8)-(5.11). Moreover, the objective value of the DP model with the solution $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ is

$$\sum_{t=1}^{T} \beta^{t-1} \left(\sum_{j\in J} o_j z^*_{j,t} + \sum_{j\in J} h_{j,t} m^*_{j,t} \right),$$

which equals to the value θ^*.

Suppose the solution $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ does not minimize the objective value of the DP model, which means there exists another solution $\{m'_{j,t}, z'_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ such that

$$\sum_{t=1}^{T} \beta^{t-1} \left(\sum_{j\in J} o_j z'_{j,t} + \sum_{j\in J} h_{j,t} m'_{j,t} \right) < \theta^*.$$

According to Proposition 1, we know $\{m'_{j,t}, z'_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ must be a feasible solution of the LP model (5.12), then the objective value of the LP model with the solution $\{m'_{j,t}, z'_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ is

$$\sum_{t=1}^{T} \beta^{t-1} \left(\sum_{j\in J} o_j z'_{j,t} + \sum_{j\in J} h_{j,t} m'_{j,t} \right) < \theta^*.$$

Therefore, the optimality of the solution $\{m^*_{j,t}, z^*_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$ in the LP model (5.12)-(5.20) is contradicted by the existence of the solution $\{m'_{j,t}, z'_{j,t}\}_{j\in J, t\in\{1,\dots,T\}}$.

□

We proved the LP model (5.12)-(5.20) can calculate the optimal purchase plan with the minimum total cost. However, the number of decision variables in the LP model is linear in $T \times (|J| + \sum_{j\in J} |\bar{B}_j|)$, which increases as the planning horizon T increases. To find a good purchase plan for the LPP-OP with longer planning horizon, we introduce a solution that minimizes the myopic cost in each period as an approximation of the optimal solution.

5.4 Myopic Solution Approximation

Depending on the length of the planning horizon T, finding the optimal solution of the long-term LP model (5.12)-(5.20) can be computationally expensive. Thus, we approximate the optimal solution by calculating the server capacity that minimizes the cost in each period without considering future periods. In the next section, we prove the myopic server purchase plan is also the optimal solution for the total cost function in some special cases.

Based on the definition of the total cost in Equation 5.7, we denote the cost in period t by function

$$H_t(n_{1,t},...,n_{|J|,t};\vec{S}_t,\vec{l}_t)=\sum_{j\in J}h_{j,t}n_{j,t}+O(n_{1,t},...,n_{|J|,t};\vec{S}_t)-\sum_{j\in J}h_{j,t}l_{j,t} \tag{5.22}$$

where $O(n_{1,t},...,n_{|J|,t};\vec{S}_t)$ is defined as the optimal solution of the operating cost model (5.1)-(5.6), and $\vec{l}_t=(l_{1,t},...,l_{|J|,t})$ is the set of servers in the cloud system at the beginning of period t, which is a constant.

We define the myopic optimal server capacity $\vec{n}_t^H=\{n_{j,t}^H\}_{j\in J}$ as the server capacity that minimizes the short-term cost function $H_t(n_{1,t},...,n_{|J|,t};\vec{S}_t,\vec{l}_t)$. We claim the myopic optimal server capacity $\vec{n}_t^H$ in period t can be obtained by solving the following LP model:

$$\min \sum_{j\in J}h_{j,t}n_j+\sum_{j\in J}o_jz_j \tag{5.23}$$

$$\text{s.t.} \sum_{j\in J}\sum_{\mathbf{v}\in\bar{B}_j}v_ix_{j\mathbf{v}}\geq s_{i,t}, \qquad \forall i\in I \tag{5.24}$$

$$z_j\leq n_j, \qquad \forall j\in J \tag{5.25}$$

$$\sum_{\mathbf{v}\in\bar{B}_j}x_{j\mathbf{v}}=z_j, \qquad \forall j\in J \tag{5.26}$$

$$z_j\in\mathbb{R}_+, \qquad \forall j\in J \tag{5.27}$$

$$x_{j\mathbf{v}}\in\mathbb{R}_+, \qquad \forall j\in J,\mathbf{v}\in\bar{B}_j \tag{5.28}$$

$$n_j\in\mathbb{R}_+. \qquad \forall j\in J \tag{5.29}$$

The objective (5.23) is to minimize the purchasing cost and the operating cost in period t. Constraints (5.24)-(5.28) are same as (5.2)-(5.6) in the operating cost model, so the service slot requirements are satisfied in period t. The additional constraint (5.29) is the domain constraint for the server capacity after the purchasing in period t, since it is a set of decision variables in the myopic problem.

Here, we prove this claim.

Theorem 5.3. *In any period $t \in \{1, ..., T\}$, the optimal solution $\{n_j^*, z_j^*\}_{j \in J}$ of the LP model (5.23)-(5.29) minimizes the short-term cost function $H_t(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t, \vec{l}_t)$ in period t, i.e. $n_j^* = n_{j,t}^H, \forall j \in J$.*

Proof. Fix an arbitrary $t \in \{1, ..., T\}$. Suppose there exist $\{n_j'\}_{j \in J} \in \mathbb{R}^{|J|}$ such that $H_t(n_1', ..., n_{|J|}'; \vec{S}_t, \vec{l}_t) < H_t(n_1^*, ..., n_{|J|}^*; \vec{S}_t, \vec{l}_t)$. Since $\{n_j'\}_{j \in J} \in \mathbb{R}^{|J|}$, then it is feasible in the LP model (5.23)-(5.29). Note that the constraints (5.24)-(5.28) are the same as the constraints in the operating cost model (5.2)-(5.6). Thus, the z_j decision variables have the same feasible region in the LP model (5.23)-(5.29) and the operating cost model (5.1)-(5.6). Therefore, the feasibility of $\{n_j'\}_{j \in J}$ in the LP model (5.23)-(5.29) implies

$$\sum_{j \in J} h_{j,t} n_j' + O(n_1', ..., n_{|J|}'; \vec{S}_t) \geq \sum_{j \in J} h_{j,t} n_j^* + \sum_{j \in J} o_j z_j^* = \sum_{j \in J} h_{j,t} n_j^* + O(n_1^*, ..., n_{|J|}^*; \vec{S}_t).$$

Note that $\sum_{j \in J} h_{j,t} l_{j,t}$ is a constant term, so

$$\begin{aligned} H_t(n_1', ..., n_{|J|}'; \vec{S}_t, \vec{l}_t) &= \sum_{j \in J} h_{j,t} n_j' + O(n_1', ..., n_{|J|}'; \vec{S}_t) - \sum_{j \in J} h_{j,t} l_{j,t} \\ &\geq \sum_{j \in J} h_{j,t} n_j^* + O(n_1^*, ..., n_{|J|}^*; \vec{S}_t) - \sum_{j \in J} h_{j,t} l_{j,t} \\ &= H_t(n_1^*, ..., n_{|J|}^*; \vec{S}_t, \vec{l}_t), \end{aligned}$$

and the contradiction forms. □

Therefore, the new LP model (5.23)-(5.29) finds the optimal server capacity $\vec{n}_t^H$ with the minimum operating cost and purchasing cost in period t, and we define the myopic purchase plan $\vec{m}_t^H = (m_{1.t}^H, ..., m_{|J|,t}^H)$ as following

$$l_{j,t} = (1 - d_j) n_{j,t-1}^H, \qquad \forall j \in J, t \in \{2, ..., T\} \tag{5.30}$$

$$m_{j,t}^H = \begin{cases} n_{j,t}^H - l_{j,t}, & n_{j,t}^H \geq l_{j,t} \\ 0, & n_{j,t}^H < l_{j,t} \end{cases} \qquad \forall j \in J, t \in \{1, ..., T\} \tag{5.31}$$

5.5 Optimal Myopic Purchasing Policy with Non-decreasing Workload

With the computational complexity reduced by solving the myopic solution instead of the long-term optimal solution, we are interested in the quality of myopic solutions

as they may lead to some additional costs in long-term. Consider an example with two classes of jobs and three types of servers. Suppose class 1 jobs are CPU intensive and class 2 jobs are memory intensive, and type 1 servers are capable for class 1 jobs only, type 2 servers are capable for class 2 jobs only, and type 3 servers can process jobs in any class with a higher price. If the jobs arrive to the cloud are mostly in class 1 in the first period and change to mostly class 2 jobs in the second period, then the myopic solution will suggest to purchase type 1 servers in first period and get type 2 servers in second period. However, the long-term optimal purchase plan is purchasing type 3 servers in the first period and operate them in the second period.

In this section, we prove the myopic server purchase plan coincides the long-term optimal server purchase plan in each period when the workload is non-decreasing. To simplify the proof, we will prove the equality of the myopic server capacity $\{\vec{n}_t^H = (n_{1,t}^H, ..., n_{|J|,t}^H)\}_{t\in\{1,...,T\}}$ and the long-term optimal server capacity $\{\vec{n}_t^* = (n_{1,t}^*, ..., n_{|J|,t}^*)\}_{t\in\{1,...,T\}}$, since the server purchase plan is the only set of decision variables that affects the server capacity. Specifically, $\vec{n}_t^H$ is the optimal solution of the myopic LP model (5.23)-(5.29) in each period t, and $\{\vec{n}_t^*\}_{t\in\{1,...,T\}}$ is the server capacity corresponds to the optimal solution $\{m_{j,t}^*, z_{j,t}^*\}_{t\in\{1,...,T\}}$ of the long-term DP model (5.8)-(5.11), where

$$\begin{aligned} n_{j,t}^* &= l_{j,t} + m_{j,t}^*, && \forall j \in J, t \in \{1, ..., T\} \\ l_{j,t} &= (1 - d_j) n_{j,t-1}^*. && \forall j \in J, t \in \{2, ..., T\} \end{aligned}$$

We define the workload is non-decreasing as follows.

Definition 5.4. *Consider the workload from period 1 to period T: $\vec{S}_1, ..., \vec{S}_T$. The workload is non-decreasing if and only if*

$$\frac{\partial H_t(n_1, ..., n_{|J|}; \vec{S}_t, \vec{l}_t)}{\partial n_j} \geq \frac{\partial H_{t+1}(n_1, ..., n_{|J|}; \vec{S}_{t+1}, \vec{l}_{t+1})}{\partial n_j} \quad \forall j \in J, t \in \{1, ..., T-1\} \tag{5.32}$$

According to the definition (see Equation 5.22), $H_t(n_1, ..., n_{|J|}; \vec{S}_t, \vec{l}_t)$ is the short-term cost in period t while operating $\vec{n} = \{n_1, ..., n_{|J|}\}$ servers with a given workload $\vec{S}_t$ and $\vec{l}_t$ installed servers. Then $\frac{\partial H_t(n_1,...,n_{|J|};\vec{S}_t,\vec{l}_t)}{\partial n_j}$ represents the marginal short-term cost of operating one more type j server in period t when the cloud is already operating $\vec{n}$ servers. Definition 5.4 states that the workload is non-decreasing if and only if the marginal cost of operating one more server from any type $j \in J$ in period t is greater than or equal to the marginal cost of adding the same server in the next period $t+1$, while the cloud having the same operating plan $\vec{n}$ in both periods. The intuition is

that the heavier workload should lead to a smaller marginal cost (a greater marginal benefit) on operating an additional server in any type. Consider the example with two classes and three types above, the class 1 jobs strongly impacts the marginal cost of type 1 servers and slightly affects the marginal cost of other types of servers. Similarly, the class 2 jobs strongly impacts the marginal cost of type 2 servers and slightly affects the marginal cost of the others. Thus, the workload shift from class 1 jobs to class 2 jobs causes the marginal cost of type 1 servers to increase, so it is not a non-decrease workload.

Definition 5.4 uses server marginal costs to focus on the overall workload among all classes instead of the class specified workload. Thus, the definition of non-decreasing overall workload is different from non-decreasing workload in every job class, i.e. $s_{i,t} \leq s_{i,t+1}, \forall i \in I, t \in \{1, ..., T-1\}$. For example, in a cloud that serves jobs in similar classes, the overall workload is still non-decreasing if the workload in one class falls but the workload from the other classes increases and covers the marginal cost loss for all servers.

We first prove the convexity of the short-term cost function $H_t(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t, \vec{l}_t)$ in all periods t which is used later in our main result.

Theorem 5.5. *For all $t \in \{1, ..., T\}$, $H_t(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t, \vec{l}_t)$ is jointly convex in $(n_{1,t}, ..., n_{|J|,t})$.*

Proof. Fix the period number $t \in \{1, ..., T\}$ arbitrarily.

First we want to show the operating cost function $O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t)$ is jointly convex in $(n_{1,t}, ..., n_{|J|,t})$ for any given workload $\vec{S}_t$.

Consider the operating cost function $O(n'_{1,t}, ..., n'_{|J|,t}; \vec{S}_t)$ with its optimal solution $\{z'_j, x'_{j\mathbf{v}}\}_{j\in J, \mathbf{v}\in \bar{B}_j}$, and another operating cost function $O(n''_{1,t}, ..., n''_{|J|,t}; \vec{S}_t)$ with its optimal solution $\{z''_j, x''_{j\mathbf{v}}\}_{j\in J, \mathbf{v}\in \bar{B}_j}$.

Fix any $0 \leq \lambda \leq 1$, then we want to show

$$\{\lambda z'_j + (1-\lambda) z''_j, \lambda x'_{j\mathbf{v}} + (1-\lambda) x''_{j\mathbf{v}}\}$$

is a feasible solution in

$$O(\lambda n'_{1,t} + (1-\lambda) n''_{1,t}, ..., \lambda n'_{|J|,t} + (1-\lambda) n''_{|J|,t}; \vec{S}_t).$$

We show that by proving

$$\{\lambda z'_j + (1-\lambda) z''_j, \lambda x'_{j\mathbf{v}} + (1-\lambda) x''_{j\mathbf{v}}\}$$

satisfies all constraints in

$$O(\lambda n'_{1,t} + (1-\lambda)n''_{1,t}, ..., \lambda n'_{|J|,t} + (1-\lambda)n''_{|J|,t}; \vec{S}_t).$$

Constraint 5.2:

$$\begin{aligned}\sum_{j\in J}\sum_{\mathbf{v}\in\bar{B}_j} v_i[\lambda x'_{j\mathbf{v}} + (1-\lambda)x''_{j\mathbf{v}}] =& \sum_{\mathbf{v}\in\bar{B}_j} v_i\lambda x'_{j\mathbf{v}} + \sum_{\mathbf{v}\in\bar{B}_j} v_i(1-\lambda)x''_{j\mathbf{v}} \\ =&\lambda\sum_{\mathbf{v}\in\bar{B}_j} v_i x'_{j\mathbf{v}} + (1-\lambda)\sum_{\mathbf{v}\in\bar{B}_j} v_i x''_{j\mathbf{v}} \\ \geq&\lambda s_{i,t} + (1-\lambda)s_{i,t} \\ =&s_{i,t}, \qquad \forall i \in I\end{aligned}$$

Constraint 5.3:

$$\lambda z'_j + (1-\lambda)z''_j \leq \lambda n'_{j,t} + (1-\lambda)n''_{j,t}, \qquad \forall j \in J$$

Constraint 5.4:

$$\begin{aligned}\sum_{\mathbf{v}\in\bar{B}_j}[\lambda x'_{j\mathbf{v}} + (1-\lambda)x''_{j\mathbf{v}}] =& \sum_{\mathbf{v}\in\bar{B}_j}\lambda x'_{j\mathbf{v}} + \sum_{\mathbf{v}\in\bar{B}_j}(1-\lambda)x''_{j\mathbf{v}} \\ =&\lambda\sum_{\mathbf{v}\in\bar{B}_j} x'_{j\mathbf{v}} + (1-\lambda)\sum_{\mathbf{v}\in\bar{B}_j} x''_{j\mathbf{v}} \\ =&\lambda z'_j + (1-\lambda)z''_j, \qquad \forall j \in J\end{aligned}$$

Constraint 5.5 and 5.6:

$$\begin{aligned}\lambda z'_j + (1-\lambda)z''_j \in \mathbb{R}^+, \qquad & \forall j \in J \\ \lambda x'_{j\mathbf{v}} + (1-\lambda)x''_{j\mathbf{v}} \in \mathbb{R}^+, \qquad & \forall j \in J, \mathbf{v} \in \bar{B}_j\end{aligned}$$

Thus, we know

$$\begin{aligned}&\sum_{j\in J} o_j[\lambda z'_j + (1-\lambda)z''_j] \\ \geq& O(\lambda n'_{1,t} + (1-\lambda)n''_{1,t}, ..., \lambda n'_{|J|,t} + (1-\lambda)n''_{|J|,t}; \vec{S}_t),\end{aligned} \tag{5.33}$$

since the operating cost function is defined as the minimum operating cost considering all feasible operating decisions (see Equation 5.1).

Now we prove the convexity of $O(\cdot;\vec{S}_t)$ directly

$$\begin{aligned}
&\lambda O(n'_{1,t},...,n'_{|J|,t};\vec{S}_t)+(1-\lambda)O(n''_{1,t},...,n''_{|J|,t};\vec{S}_t)\\
=&\lambda\sum_{j\in J}o_jz'_j+(1-\lambda)\sum_{j\in J}o_jz''_j\\
=&\sum_{j\in J}o_j[\lambda z'_j+(1-\lambda)z''_j]\\
\geq&O(\lambda n'_{1,t}+(1-\lambda)n''_{1,t},...,\lambda n'_{|J|,t}+(1-\lambda)n''_{|J|,t};\vec{S}_t). \qquad \text{by Equation 5.33}
\end{aligned}$$

Therefore, since $\sum_{j\in J}h_{j,t}n_{j,t}$ is linear and jointly convex in $(n_{1,t},...,n_{|J|,t})$, then $H_t(n_{1,t},...,n_{|J|,t};\vec{S}_t)$ is jointly convex in $(n_{1,t},...,n_{|J|,t})$. □

Now we prove the main theorem in this section: the myopic solutions are optimal when the workload is non-decreasing in all periods $t\in\{1,...,T\}$.

Theorem 5.6. *Suppose the system experiences workload $\vec{S}_1,...,\vec{S}_T$ during periods 1 to T. Let $\vec{n}_t^H$ be the myopic server capacity that minimizes the short-term cost in Equation 5.22 at each period t, and $\{\vec{n}_t^*\}_{t\in\{1,...,T\}}$ be the long-term optimal server capacity that optimizes DP model (5.8)-(5.11).*

If the workload is non-decreasing across period 1 to period T, then the myopic server capacity is equal to the long-term optimal server capacity, i.e. $\vec{n}_t^H=\vec{n}_t^*,\forall t\in\{1,...,T\}$.

Proof. We prove the theorem in two stages:

(i) If the overall workload is non-decreasing with time, then the myopic server capacity is non-decreasing, i.e. $n_{j,t}^H\leq n_{j,t+1}^H,\forall j\in J,t\in\{1,...,T-1\}$.

(ii) If $n_{j,t}^H\leq n_{j,t+1}^H,\forall j\in J,t\in\{1,...,T-1\}$, then the myopic server capacity is also optimal for the total cost function, i.e. $\vec{n}_t^H=\vec{n}_t^*,\forall t\in\{1,...,T\}$.

Proof of Part (i): Fix an arbitrary $t\in\{1,...,T-1\}$. Assume the workload is non-decreasing, then we know $\frac{\partial H_t(n_1,...,n_{|J|};\vec{S}_t,\vec{l}_t)}{\partial n_j}\geq\frac{\partial H_t(n_1,...,n_{|J|};\vec{S}_{t+1},\vec{l}_{t+1})}{\partial n_j}$ by Definition 5.4.

The first order condition states

$$\lim_{n_j\to n_{j,t}^H}\frac{\partial H_t(n_{1,t}^H,...,n_j,n_{j+1,t}^H,...,n_{|J|,t}^H;\vec{S}_t,\vec{l}_t)}{\partial n_j}=0,\quad\forall j\in J$$

and

$$\lim_{n_j\to n_{j,t+1}^H}\frac{\partial H_t(n_{1,t+1}^H,...,n_j,n_{j+1,t+1}^H,...,n_{|J|,t+1}^H;\vec{S}_{t+1},\vec{l}_{t+1})}{\partial n_j}=0.\quad\forall j\in J$$

According to Theorem 5.5, we know that

$$n_{j,t}^H \leq n_{j,t+1}^H, \forall j \in J$$

based on the convexity of the short-term cost function $H_t(\cdot)$.

Proof of Part (ii): We use the similar arguments as the proof of the Proposition 3-1 in Heyman and Sobel's work [29].

Note that the objective function of the DP model (5.8) can be rewritten as the summation of the short-term cost in each period t

$$\sum_{t=1}^{T} \beta^{t-1} \left(O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t) + \sum_{j \in J} h_{j,t} m_{j,t} \right) \tag{5.34}$$

$$= \sum_{t=1}^{T} \beta^{t-1} \left(\sum_{j \in J} h_{j,t} n_{j,t} + O(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t) - \sum_{j \in J} h_{j,t} l_{j,t} \right) \tag{5.35}$$

$$= \sum_{t=1}^{T} \beta^{t-1} H_t(n_{1,t}, ..., n_{|J|,t}; \vec{S}_t, \vec{l}_t). \tag{5.36}$$

Further, note that the myopic server capacity $\vec{n}_t^H$ minimizes the short-term cost in each period t, which means for any server capacity $\vec{n}_t \in \mathbb{R}_+^{|J|}$

$$H_t(\vec{n}_t^H; \vec{S}_t, \vec{l}_t) \leq H_t(\vec{n}_t; \vec{S}_t, \vec{l}_t). \qquad \forall t \in \{1, ..., T\}$$

Thus, if the myopic server capacity $\{\vec{n}_t^H\}_{t \in \{1,...,T\}}$ is feasible in the DP model (5.8)-(5.11), then it is the optimal solution, since the objective function of the DP model is a weighted sum of independent cost functions (Equation 5.36).

We want to prove the feasibility of $\{\vec{n}_t^H\}_{t \in \{1,...,T\}}$ in the DP model. Notice the constraints (5.24)-(5.26) and (5.27)-(5.28) are the same as the constraints in the operating cost model (5.1)-(5.6). Then $\{\vec{n}_t^H\}$ is feasible in the operating cost model (5.1)-(5.6) at any period t. Now we want to show $m_{j,t} \geq 0, \forall j \in J, t \in \{1, ..., T\}$. Recall

$$n_{j,t} = l_{j,t} + m_{j,t}, \qquad \forall j \in J, t \in \{1, ..., T\}$$
$$l_{j,t} = (1 - d_j) n_{j,t-1}, \qquad \forall j \in J, t \in \{2, ..., T\}$$

which implies

$$m_{j,t} = n_{j,t} - (1 - d_j) n_{j,t-1}. \qquad \forall j \in J, t \in \{2, ..., T\}$$

Notice the constraint $m_{j,t} \geq 0$ is satisfied by the myopic policy if and only if $n_{j,t+1}^H \geq$

$(1-d_j)n^H_{j,t}, \forall j \in J, t \in \{1,...,T-1\}$. By assumption, we know

$$n^H_{j,t+1} \geq n^H_{j,t} \geq (1-d_j)n^H_{j,t}. \qquad \forall 0 \leq d_j \leq 1, j \in J, t \in \{1,...,T-1\}$$

Therefore, if the workload is non-decreasing, then statements (i) and (ii) imply the myopic policy is a feasible and optimal policy for the long-term cost function. □

In practice, solving the individual myopic optimal solutions can save a lot of processing time compared to solving the entire LPP-OP with an LP model (5.12)-(5.20). We show the benefit of the myopic models in our experiments. Furthermore, if the workload is decreasing in some period, then the myopic solution can still be used as an approximation of the long-term optimal solution, since the myopic solution is always feasible for the long-term LP model (5.12)-(5.20).

5.6 Experimental Results

The experiments in this chapter have two main goals:

1. Investigate the run-time advantage of solving the myopic models instead of the long-term LP model, and also do a sanity check on the optimality of the myopic solution with non-decreasing workload.

2. Test the quality of the myopic solution compared to the long-term optimal solution when the workload is decreasing in some periods.

To investigate the algorithm behaviour with different problem sizes, we generated problem instances that have 20 job classes with 20, 30, 40, and 50 server types. Each problem size has three instances, and every instance is solved by the full LP model and the myopic model with a planning horizon of 10, 20, and 40 periods. The problem instances with 10, 20, and 40 periods have a discount factor of 0.65, 0.8, and 0.9 respectively, so the costs after the planning horizon have a weight less than 0.015.

We used Gurobi 9.5.1 to solve the full LP models (5.12)-(5.20) and the myopic LP models (5.23)-(5.29). Each full LP model is given 100GB of memory and executed with a time limit of 3600 seconds (1 hour). The myopic LP model for any period in a problem instance is given 100GB of memory and executed with a time limit of 300 seconds (5 minutes). The full LP model has longer time limit but the same memory as each myopic LP model, because the processing time of the myopic models for different periods accumulates, but the memory usage of the myopic LP model for the previous period is released when building the next myopic LP model. This is another advantage of solving the problem with myopic LP models.

Table 5.2: Run-time Comparison of Different LP models with Non-decreasing Workload

Problem Sizes			Full LP Model	Myopic LP Models
# classes	# types	# periods	Run-time(s)	Total Run-time(s)
20	20	10	85.59	**54.63**
20	20	20	191.86	**106.73**
20	20	40	386.53	**210.62**
20	30	10	121.60	**79.04**
20	30	20	274.43	**155.58**
20	30	40	581.64	**311.94**
20	40	10	196.49	**120.86**
20	40	20	448.78	**241.76**
20	40	40	1037.62	**482.49**
20	50	10	220.05	**134.39**
20	50	20	472.43	**266.54**
20	50	40	MemOut	**522.17**

When the workload is non-decreasing in all job classes, Table 5.2 shows the run-times of the two different LP models (full and myopic). In all problem instances, the myopic LP model shows an advantage in processing time, especially as the number of periods increases. Recall the myopic model solves T LP models, where each model has the number of decision variables linear to the number of job classes times the number of server types ($|I| \times |J|$). Thus, the computational complexity of the myopic LP model is linear in T and exponential in $|I| \times |J|$. In contrast, the number of the decision variables in the full LP model is linear in $T \times |I| \times |J|$, which expands the search space with a size exponential in $T \times |I| \times |J|$.

Other than the run-time, Table 5.2 uses "MemOut" to indicate the scenario that the LP model could not be implemented within the memory limit. Note that memory issues only occurred on the Full LP model in our test cases. This observation supports the advantage of the myopic LP model on the memory usage, since the algorithm only processes the myopic LP model of one period at a time, and its memory is released after the model of each period is solved. In all problem instances that the full LP model could solve, the solution of the myopic LP model is the same as the solution of the full LP model when the workload is non-decreasing. This observation is consistent with the Theorem 5.6 in Section 5.5.

Despite the results above, cloud systems may experience declining workload in some job classes. In this case, Theorem 5.6 cannot guarantee the myopic solution is optimal in the long-term. We would like to know the quality of the myopic solution as an approximation of the long-term optimal solution.

In the following experiment, we kept the same settings on the problem instances with the exception of changing the workload in each period. Instead of the non-

Figure 5.2: The generated service slots requirement of two job classes with non-monotonic decreasing workload in 20 consecutive periods.

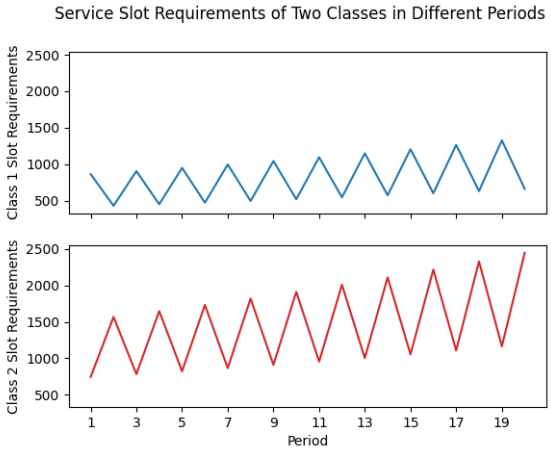


Figure 5.3: The generated service slots requirement of two job classes with non-monotonic increasing workload in 20 consecutive periods.

decreasing workload in all classes, we set the workload of each job class in all periods to be alternatively increasing and decreasing: half of the classes have their workload increases after odd periods, and the other half of the classes have their workload increases after even periods. For example, Figure 5.2 shows two classes with alternating workload and the aggregate workload is decreasing overtime, we call this workload behaviour "non-monotonic decreasing". Analogously, Figure 5.3 shows two classes with "non-monotonic increasing" workload.

We want to test the difference between the solutions from the full LP model and the myopic LP model with the workload settings above. Table 5.3 shows the results from the models with non-monotonic decreasing workload. In all problem instances, the myopic LP model has an advantage on the processing time compared to the full LP model. When the full LP model has enough memory, it always finds the optimal

Table 5.3: Run-time and Solution Value Comparison of Different LP models with Non-monotonic Decreasing Workload

Problem Sizes			Full LP Model	Myopic LP Models	
# classes	# types	# periods	Run-time(s)	Total Run-time(s)	Optimality Gap
20	20	10	94.57	**53.15**	0.73%
20	20	20	205.92	**105.83**	0.91%
20	20	40	383.36	**206.02**	1.16%
20	30	10	134.73	**80.46**	1.58%
20	30	20	337.49	**155.21**	0.84%
20	30	40	741.83	**308.39**	0.96%
20	40	10	215.29	**124.94**	0.87%
20	40	20	581.05	**242.70**	0.61%
20	40	40	1331.40	**471.27**	1.32%
20	50	10	249.77	**136.21**	0.69%
20	50	20	603.42	**265.13**	0.64%
20	50	40	MemOut	**522.79**	N/A

Table 5.4: Run-time and Solution Value Comparison of Different LP models with Non-monotonic Increasing Workload

Problem Sizes			Full LP Model	Myopic LP Models	
# classes	# types	# periods	Run-time(s)	Total Run-time(s)	Optimality Gap
20	20	10	91.22	**54.64**	2.01%
20	20	20	219.57	**107.31**	2.96%
20	20	40	450.51	**203.55**	4.62%
20	30	10	151.48	**81.94**	1.52%
20	30	20	332.83	**156.80**	3.14%
20	30	40	664.16	**309.14**	3.28%
20	40	10	224.69	**124.53**	2.48%
20	40	20	508.21	**245.46**	3.29%
20	40	40	1208.30	**476.49**	4.41%
20	50	10	243.52	**134.84**	1.35%
20	50	20	540.32	**264.46**	2.66%
20	50	40	MemOut	**514.09**	N/A

solution within the time limit in our tests, so the optimality gap is calculated by

$$\frac{\text{Myopic Solution} - \text{Optimal Solution}}{\text{Optimal Solution}}$$

where the myopic solution is the solution of the myopic LP model. In the largest problem of our experiment, the memory runs out for the full LP model. As a result, the model does not give us any lower bound on the solution and the optimality gap of the myopic solution cannot be calculated. Similar to the experimental results with the non-decreasing workload, the run-time of the full LP model grows faster compared to the run-time of the myopic LP model as the planning horizon increases. However, the solution from the myopic LP model is no longer optimal without non-decreasing workload, and across all of our test cases, the optimality gap of the myopic solution is always less or equal to 2%.

Table 5.4 shows the results from the full LP model and the myopic LP model with non-monotonic increasing workload. These results also show the runtime advantage on the myopic LP model, but the optimality gap of the myopic solutions is 2-3% greater than the results in Table 5.3. We observe both the full model and the myopic model stop purchasing server in the later periods when the workload is non-monotonic decreasing, since there are enough servers in the system. Thus, the myopic LP model only has extra purchasing cost in early periods. When the workload is non-monotonic increasing, both the full model and the myopic model purchase servers in all periods, and the myopic model tends to purchase extra servers through the entire planning horizon with additional costs. Therefore, the myopic model makes more mistakes and has a larger optimality gap when the workload is non-monotonic increasing.

We observe that the optimality gap of the myopic solution is not always increasing as the number of periods increases in Table 5.3 and Table 5.4. Intuitively, we expect the myopic model performs worse on the problems with longer planning horizon, but this is not what we observe due to the following reason. As the planning horizon increases, the optimal total cost increases. Thus, if the myopic model makes few mistakes in later periods, then the extra costs caused by the myopic solution in the early periods is amortized over more periods resulting in a smaller percentage gap.

5.7 Conclusion

In this chapter, we defined the long-term purchase plan optimization problem (LPP-OP) to find the best server purchase plans for cloud systems, and introduced a linear programming (LP) model to solve the problem. To simplify the solution process and

solve for problems with larger size, we designed a myopic approximation strategy of the LP model, which solves a set of smaller LP models that each optimizes the cost in a single period. We proved the myopic approximation is the long-term optimal server purchase plan when the workload is non-decreasing. With this theorem, we can solve the server purchase plan optimization problem that only considers the current cost in each period, as long as the market is growing. The myopic method also removes the error from the data forecasting in our deterministic framework, since it only uses the parameters in the current period. However, a fundamental assumption required for the optimal myopic solution is that the future demand is independent of the current server capacity in the cloud. The correctness of this assumption requires additional studies or surveys.

Our experimental results support the advantage of the myopic approximation in the runtime complexity and the memory complexity compared to the full LP model. We have a sanity check on the optimality of the myopic solution while the workload is non-decreasing in our experiment. When the workload in some job classes is decreasing in some periods, the experimental results show the myopic approximation has a higher accuracy when the overall workload is decreasing in long-term. The intuition behind is that if the myopic approximation makes mistakes on the server purchasing plans, then less approximation error is made when there is less purchasing cost. The operating cost takes a higher percentage in the total cost of the cloud with decreasing workload, since the cloud requires fewer servers compared to the cloud with increasing workload.

For future work, an interesting study would be formulating an approach between the myopic solution and the long-term optimal solution. A possible implementation is to split the entire planning horizon into multiple groups of consecutive periods, and calculate the optimal purchase plans in each group. Such an approach seeks to balance the computational complexity and the solution quality. The choice of the period groups can be made depending on the workload trend in order to increase the accuracy of the approximation. Another way to extend the myopic approximation is to solve the LP model with rolling horizon. With a rolling horizon of length K, the purchasing plan in each period t is calculated by the LP model with the periods $\{t, ..., t + K\}$ as the planning horizon. Our myopic approximation strategy is the example of solving the LP model with the length 1 rolling horizon.

Our framework may apply to similar systems other than the cloud. For example, health care systems make periodic plans for the surgeries while considering the available recovery wards [4]. Each ward may support multiple patients after different types of surgery, which is similar to the job-to-server allocation problem. However,

the details of the model require more study based on the restrictions in the health care system.

Chapter 6

Conclusion

This chapter summarizes the results and contributions of the previous chapters, and discusses some directions for future work.

6.1 Summary of Contributions

In this thesis, we formally defined the short-term and long-term server capacity planning problems in infrastructure as a service (IaaS) clouds based on the real cloud structure. In practice, a cloud receives multiple classes of jobs with stochastic arrival and service time and must assign them to heterogeneous servers. Thus, two major challenges of the server capacity planning problems are: calculating a deterministic server capacity to satisfy the stochastic workload and assigning the multi-class jobs to heterogeneous servers. We applied combinatorial optimization models to solve both the short-term and long-term server capacity planning problem, and we hybridized the combinatorial model with queueing models to consider the stochastic behaviors in the cloud. Finally, we showed the usefulness of the frameworks with some empirical results.

In Chapter 3, we introduced the hybrid framework based on the combinatorial optimization models and queueing models to resolve the two challenges in the problem. Queueing models are used to calculate the resource requirements with respect to the stochastic workload, and the combinatorial models allocate the resources to a set of servers with the minimum operating cost. The numerical results including a case study of the Google Cloud show the hybrid framework can solve the short-term server capacity planning problem in real cloud in a reasonable time. From the case study with the dataset from Google Cloud, however, we observed some weaknesses of the problem definition and the framework. Some of the problem assumptions are too strong for real cloud systems. For example, the problem assumes that the cloud

experiences a workload with stable job arrival rates across days or weeks. In practice, the cloud has different workload in different days, and a few hours in the day are much busier compared to the rest of the hours. This violation on our assumptions causes our framework to underestimate the resource usage of the cloud, and the framework produces the solution that fails to satisfy the SLAs during the busy hours.

In our experiments, we also compared different queueing models, and the results indicate the $GI/GI/n$ queues fit better on the cloud with heavy workload and non-exponential interarrival or service time compared to the $M/M/n$ queues. The results also show that a linear programming model with rounding strategy solves the allocation problem close to the optimal integral solution in a shorter time compared to the integer programming model.

Chapter 5 extends the server capacity planning problem to a long-term basis, which involves optimizing the periodic server purchasing plan with respect to the changing workload. We introduced a linear programming (LP) model to optimize the server purchasing plan with an objective of minimizing the operating costs and purchasing costs. However, a longer planning horizon causes the size of the LP model to increase, as well as its computational complexity and memory requirement. Thus, we develop a myopic approximation method for the LP model that decomposes the model into submodels per time period, where each submodel requires much less computation and memory. Solving the submodels sequentially is shown to have a runtime and memory advantage compared to solving the full LP model in the experiments, and we prove the myopic solution is the optimal solution when the workload is non-decreasing across the entire planning horizon. The numerical results support our theorem, and present the usefulness of the LP model on small clouds and the myopic approximation method on large clouds. Unfortunately, the myopic approximation might be suboptimal when the workload decreases in some periods, and we could not prove a good theoretical bound on the optimality gap of the myopic approximation. However, the optimality gap of the myopic approximation is acceptable ($< 5\%$) when the workload is not non-decreasing in our experiment. Suppose the workload is not always non-decreasing (e.g. the workload increases and decreases alternatively), the results show the quality of the myopic approximation is worse when the workload has an upward trend compared to a downward trend in the long-term. The reason is that the myopic solutions make mistakes on the server purchasing plans, and more servers are purchased when the workload has an upward trend instead of downward. The myopic model correctly calculates the optimal operating cost, and the operating cost is the major cost when the workload has a downward trend.

6.2 Future Work

In Chapter 4, we observed that some assumptions are violated by the real cloud dataset. One of the major assumptions is that the workload of the jobs does not change over time in the short-term planning problem. In future work, the assumption can be removed by using time varying queues (e.g. $G_t/G_t/n_t$) to calculate the time varying service slot requirements $s_i(t)$ for all job class $i \in I$. The performance of the time varying queues is studied by Pender and Ko [46], and they provide a promising approximation on the probability of excessive delay in the time varying queues (see Section 4.3 in Pender and Ko [46] for details). Therefore, we can discretize the time horizon into a set of time points $T = \{t_1, t_2, ..., t_n\}$, and then use the binary search algorithm mentioned in Section 3.3.2 to calculate the service slot requirements $\{s_i(t)\}_{i \in I}$ in all time points $t \in T$.

The integer programming (IP) model (3.2)-(3.6) introduced in Chapter 3 considers only one set of service slot requirements, so the model needs to be reformulated. With the goal of calculating the cheapest set of servers that satisfies the workload in all time points, we focus on the time where the workload peak arrives. The time of peak workload is obtained by the following:

$$\arg\max_{t \in T} \left(\alpha \sum_{i \in I} s_i(t) c_i + \beta \sum_{i \in I} s_i(t) m_i \right),$$

where c_i and m_i are the CPU and memory requirement of class $i \in I$ jobs, and the $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ are weights for the CPU and memory requirements, respectively. We obtain multiple time points $\bar{T} = \{t_1, ..., t_m\} \subseteq T$ with peak workload by different α and β settings depending on the cloud environment. Then the IP model that considers the peak workload is formulated as the following.

Decision Variables:

z_j The number of operating servers of type j through the entire time horizon, $\forall j \in J$

$x_{j\mathbf{v}t}$ The number of servers in type j that operate in configuration $\mathbf{v}$ at time point t. $\forall j \in J, \mathbf{v} \in \bar{B}_j, t \in \bar{T}$

IP Model:

$$\min \sum_{j \in J} O_j z_j \tag{6.1}$$

$$\text{s.t.} \sum_{j \in J} \sum_{\mathbf{v} \in \bar{B}_j} v_i x_{j\mathbf{v}t} \geq s_i(t), \qquad \forall i \in I, t \in \bar{T} \tag{6.2}$$

$$\sum_{\mathbf{v} \in \bar{B}_j} x_{j\mathbf{v}t} \leq z_j, \qquad \forall j \in J, t \in \bar{T} \tag{6.3}$$

$$z_j \in \mathbb{Z}_+, \qquad \forall j \in J \tag{6.4}$$

$$x_{j\mathbf{v}t} \in \mathbb{Z}_+. \qquad \forall j \in J, \mathbf{v} \in \bar{B}_j, t \in \bar{T} \tag{6.5}$$

The objective (6.1) of the IP model is to minimize the operating cost of the servers. Constraints (6.2) force the servers to satisfy the workload at all time $t \in \bar{T}$ with different configurations. Constraints (6.3) restrict the jobs to be assigned on operating servers only. Constraints (6.4)-(6.5) are the domain constraints that limit the number of operating servers and the number of servers taking each configuration to be positive integers.

The time varying workload IP model (6.1)-(6.5) is about $|\bar{T}|$ times larger than the IP model (3.2)-(3.6), since the time varying model has to determine the configurations of the same set of servers to satisfy multiple workload. To reduce the computational complexity, we can apply Benders decomposition [6] to the IP model (6.1)-(6.5). A naive approach is to initialize the master problem (MP) as:

$$\min \sum_{j \in J} O_j z_j \tag{6.6}$$

$$\text{s.t.} \sum_{\mathbf{v} \in \bar{B}_j} x_{j\mathbf{v}t} \leq z_j, \qquad \forall j \in J, t \in \bar{T} \tag{6.7}$$

$$z_j \in \mathbb{Z}_+, \qquad \forall j \in J \tag{6.8}$$

$$x_{j\mathbf{v}t} \in \mathbb{Z}_+, \qquad \forall j \in J, \mathbf{v} \in \bar{B}_j, t \in \bar{T} \tag{6.9}$$

which is the original IP model (6.1)-(6.5) without the workload information (6.2). The workload satisfaction problem is decomposed into $|\bar{T}|$ subproblems $P = \{P_1, P_2, ..., P_{|\bar{T}|}\}$. After the MP produces an optimal solution $\{\hat{z}_j, \hat{x}_{j\mathbf{v}t}\}_{j \in J, \mathbf{v} \in \bar{B}_j, t \in \bar{T}}$, each subproblem $P_t, t \in \bar{T}$ checks the feasibility of the configurations $\{\hat{x}_{j\mathbf{v}t}\}_{j \in J, \mathbf{v} \in \bar{B}_j}$ in the following constraints,

$$\sum_{j \in J} \sum_{\mathbf{v} \in \bar{B}_j} v_i x_{j\mathbf{v}t} \geq s_i(t), \forall i \in I.$$

If the configurations are infeasible, then one of the unsatisfied workload constraints is

added to the MP. The solution of the MP will converge to the optimal server plan by solving the MP and the subproblems iteratively. According to the designer's choice, the algorithm may solve multiple subproblems and add multiple constraints to the MP in each iteration.

Another approach to reduce the computational complexity is relaxing the integer variables in the IP model to continuous variables. The rounding strategies presented in Section 3.3.3 are also applicable to the time varying problem, and the results in Chapter 3 show the guaranteed feasibility and the accuracy of the rounded solution.

6.3 Conclusion

This thesis formalizes the server capacity planning problem of the cloud with stochastic heterogeneous customer requests and deterministic multi-type server capacity. We introduce an idea of virtualizing the resources allocated for the jobs as service slots, then develop a framework that hybridizes queueing models and combinatorial optimization models based on service slots to solve the problem. We extended the planning problem to consider the server purchasing plan in the long-term, and adapted the combinatorial models to solve the new problem. We believe the hybrid framework can be applied to other scheduling problems with stochastic workload, such as hospital management [4] and public transportation management [30, 10].

Appendix A

Waiting Time Distribution Approximation of $GI/GI/n$ Queues

Whitt introduced some approximations on the waiting time distributions in $GI/GI/n$ queues [67]. We choose to use the heavy traffic approximation in this work, since the cloud is a large system that satisfies the assumptions on heavy traffic, and also this approximation provides a simple calculation on the waiting time distribution.

Suppose the mean arrival rate of the jobs in a $GI/GI/n$ queue is λ, and the mean lifetime of the jobs in this queue is $\frac{1}{\mu}$. Then the distribution of the random variable W that represents the waiting time of the jobs in this queue can be approximated as follows

$$P(W > t) \approx \alpha e^{-\eta\mu t}$$

where,

$$\begin{aligned}
\rho &= \frac{\lambda}{\mu} \\
\eta &\approx \frac{2(n-\rho)}{C(A)^2 + C(L)^2} \\
\alpha &\approx \eta E(W).
\end{aligned}$$

In the chapter 2.4 of Whitt [67], a heavy traffic approximation on the expected waiting time of the jobs in a $GI/GI/n$ queue was also studied

$$E(W) \approx \frac{C(A)^2 + C(L)^2}{2} \cdot E(W_{M/M/n}),$$

where $E(W_{M/M/n})$ is expected waiting time of the jobs in an $M/M/n$ queue with the same mean arrival rate and mean lifetime.

The expected waiting time of the jobs in an $M/M/n$ queue is well studied and can

be calculated by

$$E(W_{M/M/n}) = \frac{1}{\mu(n-\rho)} \cdot \gamma(\rho, n),$$

where the function $\gamma(\cdot,\cdot)$ is defined in Equation 2.3.

Thus, we can simplify the calculations and the simplified approximation becomes

$$P(W > t) \approx \frac{1}{\mu} \cdot \gamma(\rho, n) \exp\left(-\mu t(n-\rho) \cdot \frac{2}{C(A)^2 + C(L)^2}\right).$$

Appendix B

Queueing Experiment Graphs

In Chapter 3.4.1, we showed the experimental results of one specific parameter setting on a set of queueing instances. Here, we present the results from the queueing models with 0.5 second expected inter-arrival time and different expected lifetimes of 40, 80, 120, or 160 seconds. The standard deviation of the inter-arrival time and the lifetime is sd times the expected inter-arrival time and the expected lifetime respectively, where sd is varied as 2, 4, or 8.

In each of the following graphs, we show the service slot requirements calculated by different models as the service level agreement (SLA) changes. The three sets of solutions are obtained from the $M/M/n$ queueing model, $GI/GI/n$ queueing model, and a $GI/GI/n$ queue based simulation method introduced in Chapter 3.4.1. The grey horizontal lines represent the offered load (i.e. $\frac{\text{Mean Lifetime}}{\text{Mean Inter-arrival Time}}$) in each setting.

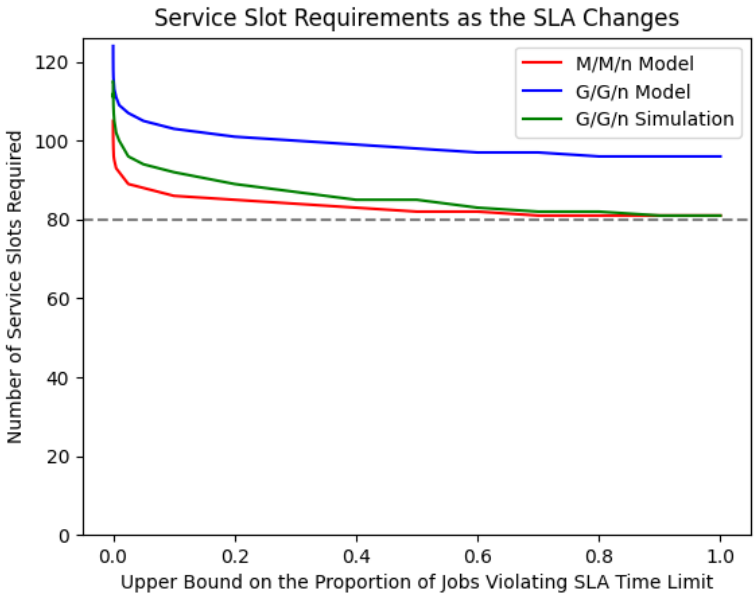


Figure B.1: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 40 seconds, and $sd = 2$.

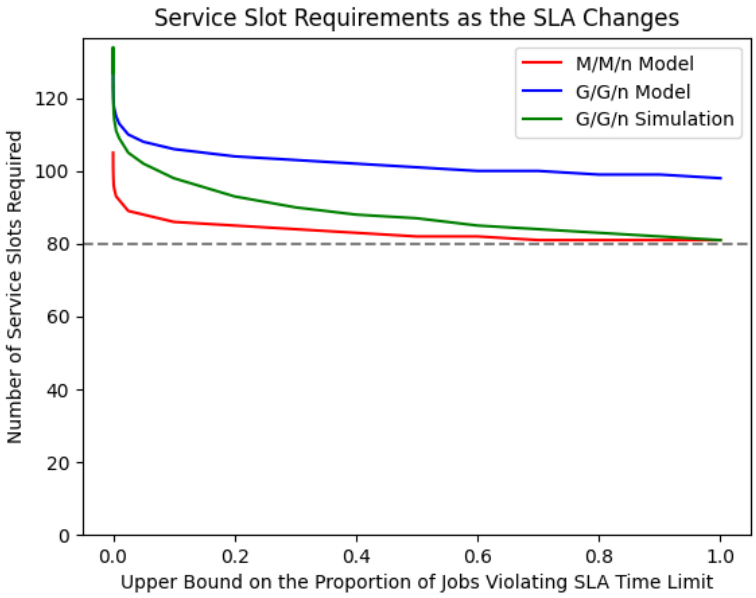


Figure B.2: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 40 seconds, and $sd = 4$.

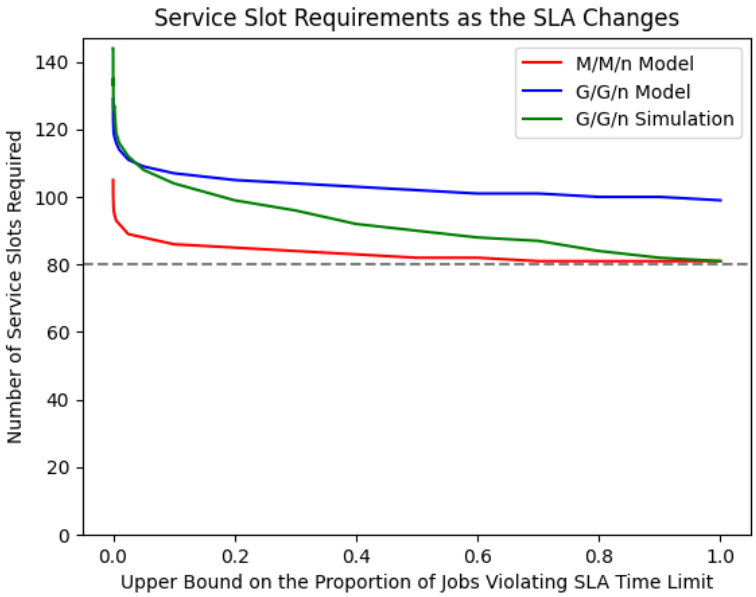


Figure B.3: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 40 seconds, and $sd = 8$.

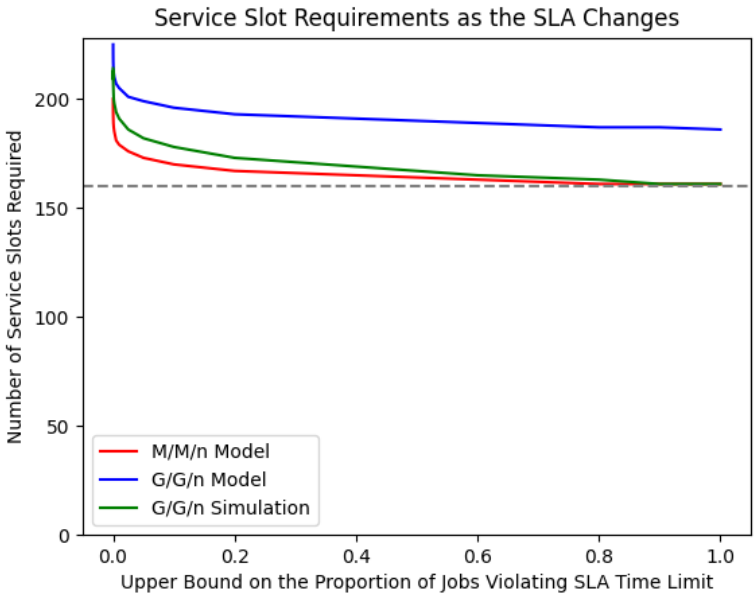


Figure B.4: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 80 seconds, and $sd = 2$.

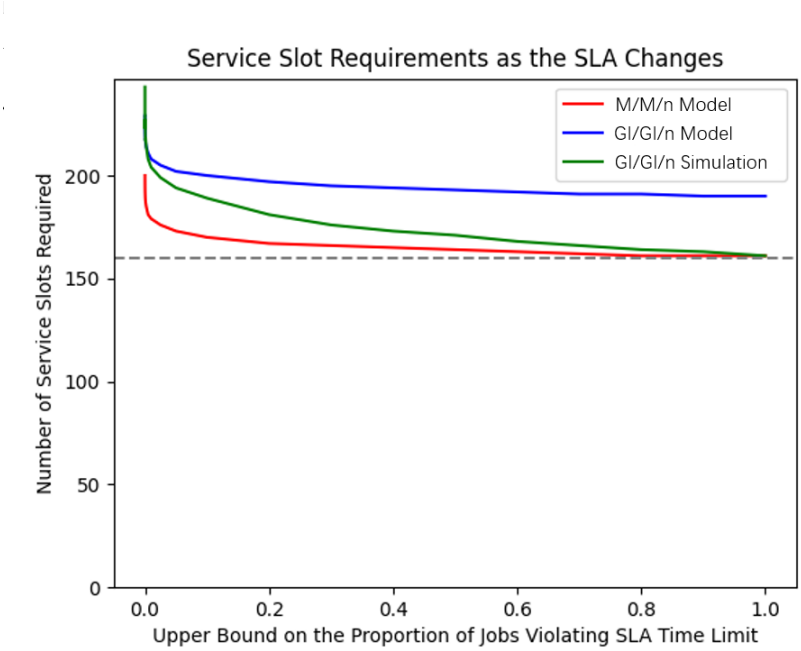


Figure B.5: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 80 seconds, and $sd = 4$.

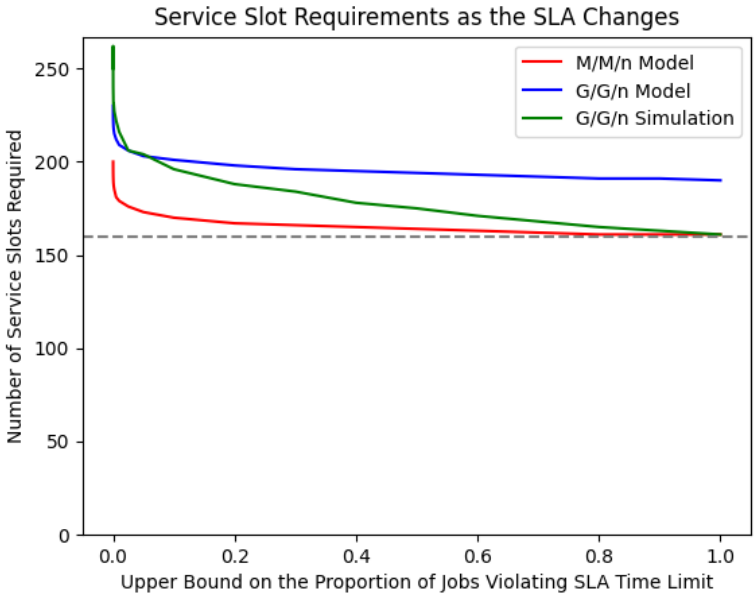


Figure B.6: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 80 seconds, and $sd = 8$.

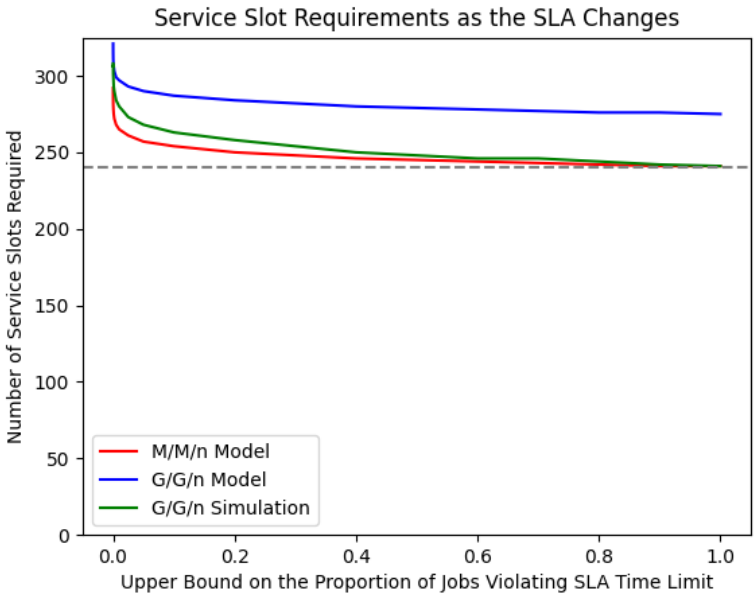


Figure B.7: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 120 seconds, and $sd = 2$.

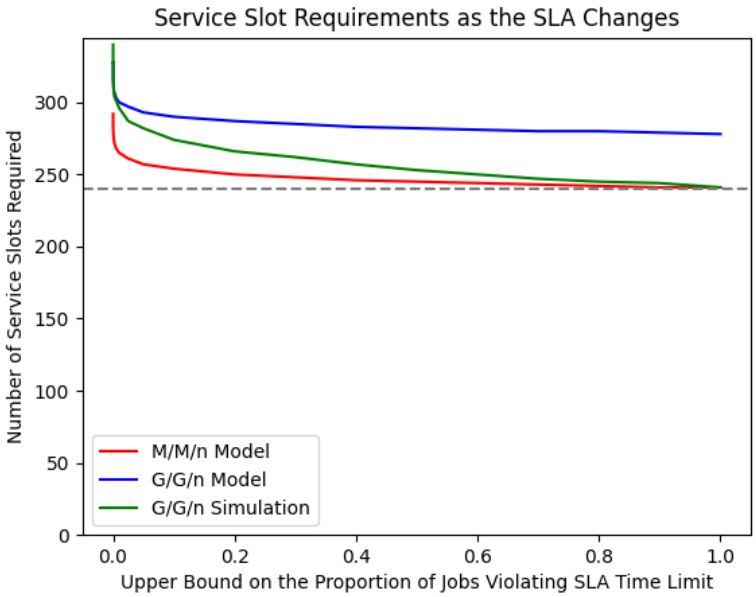


Figure B.8: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 120 seconds, and $sd = 4$.

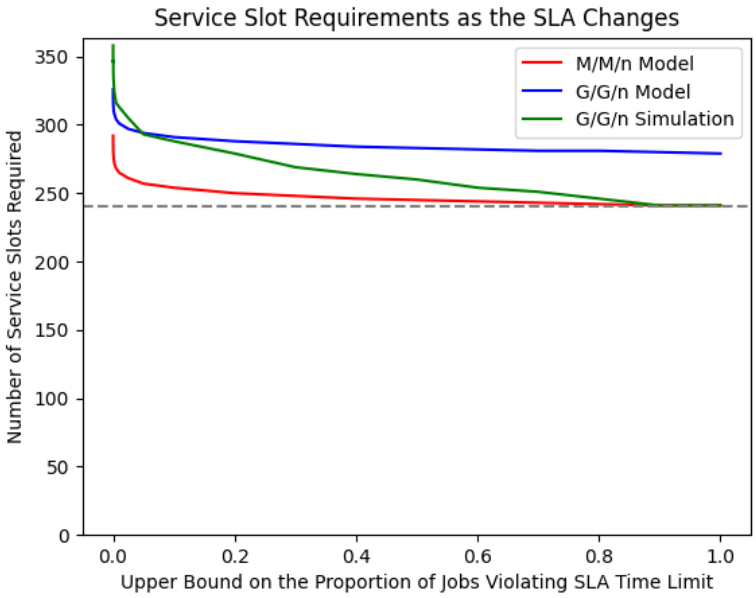


Figure B.9: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 120 seconds, and $sd = 8$.

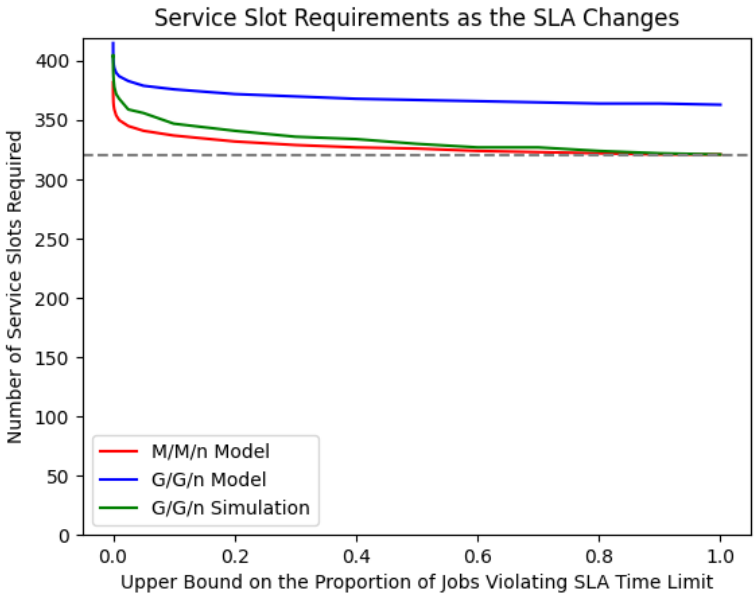


Figure B.10: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 160 seconds, and $sd = 2$.

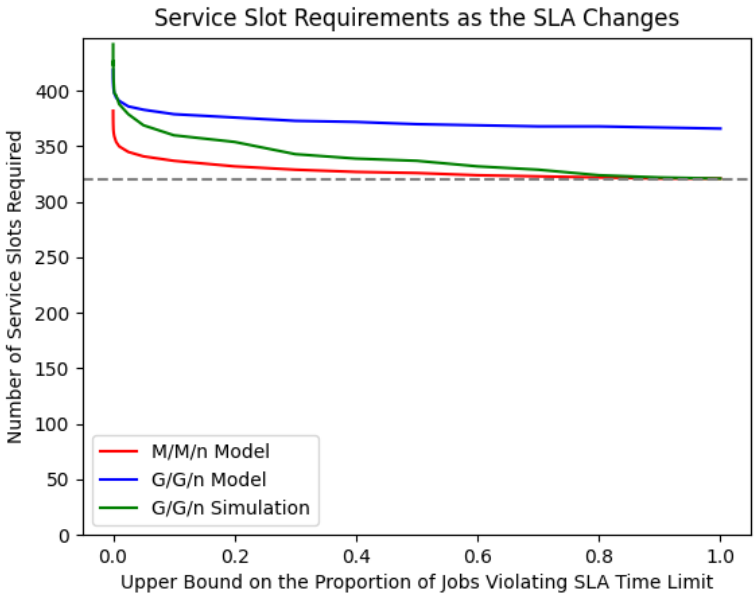


Figure B.11: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 160 seconds, and $sd = 4$.

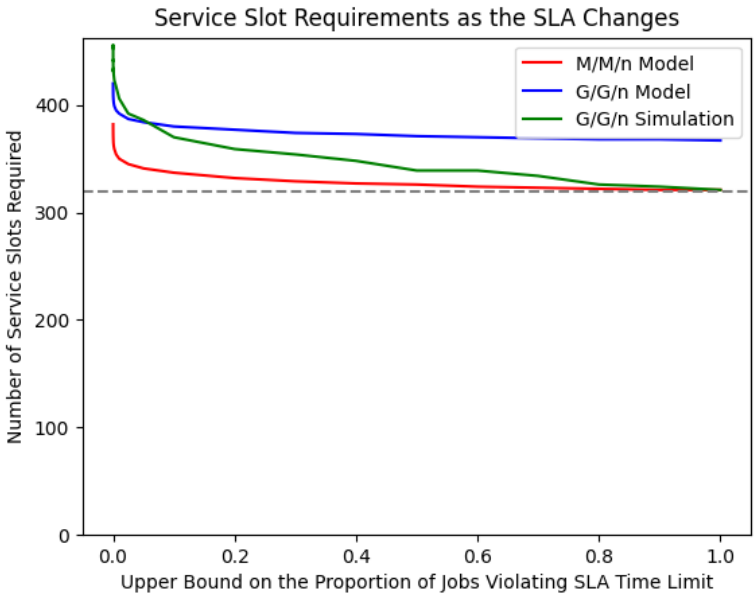


Figure B.12: The number of service slots required in the queues calculated by different models as the upper bound on the probability of a job waits for more than 10 seconds changes. The queues have expected lifetime of 160 seconds, and $sd = 8$.

Bibliography

[1] Majid Aarabi and Sajedeh Hasanian. "Capacity planning and control: a review". In: *International Journal of Scientific & Engineering Research* 5.8 (2014), pp. 975–984.

[2] Tobias Achterberg. "SCIP: solving constraint integer programs". In: *Mathematical Programming Computation*. Vol. 1. 2009, pp. 1–41. DOI: 10.1007/s12532-008-0001-1.

[3] Fares Alharbi, Yu-Chu Tian, Maolin Tang, Wei-Zhe Zhang, Chen Peng, and Minrui Fei. "An Ant Colony System for energy-efficient dynamic Virtual Machine Placement in data centers". In: *Expert Systems with Applications* 120 (2019), pp. 228–238. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.11.029. URL: https://www.sciencedirect.com/science/article/pii/S0957417418307498.

[4] Edilson Arruda. "Long-Term Integrated Surgery Room Optimization and Recovery Ward Planning, with a Case Study in the Brazilian National Institute of Traumatology and Orthopedics (INTO)". In: *European Journal of Operational Research* 264 (Feb. 2018), pp. 870–883. DOI: 10.1016/j.ejor.2016.09.021.

[5] Nils Arne Bakke and Roland Hellberg. "The challenges of capacity planning". In: *International journal of production economics* 30 (1993), pp. 243–264.

[6] J.F. Benders. "Partitioning procedures for solving mixed-variables programming problems". In: *Numer. Math.* 4.1 (1962), pp. 238–252. DOI: 10.1007/BF01386316.

[7] Eshetie Berhan. "Bank service performance improvements using multi-sever queue system". In: *IOSR Journal of Business and Management* 17.6 (2015), pp. 65–69.

[8] Mark Brinda and Michael Heric. "The changing faces of the cloud". In: *Bain Company* (2017).

[9] J. Carlstrom and R. Rom. "Application-aware admission control and scheduling in Web servers". In: *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 2. 2002, 506–515 vol.2. DOI: 10.1109/INFCOM.2002.1019295.

[10] Avishai Ceder. *Public transit planning and operation: Modeling, practice and behavior*. CRC press, 2016.

[11] Xiaolin Chang, Bin Wang, Jogesh K. Muppala, and Jiqiang Liu. "Modeling Active Virtual Machines on IaaS Clouds Using an M/G/m/m+K Queue". In: *IEEE Transactions on Services Computing* 9.3 (2016), pp. 408–420. DOI: 10.1109/TSC.2014.2376563.

[12] Xiangping Chen, Prasant Mohapatra, and Huamin Chen. "An Admission Control Scheme for Predictable Server Response Time for Web Accesses". In: *Proceedings of the 10th International Conference on World Wide Web*. WWW '01. Hong Kong, Hong Kong: Association for Computing Machinery, 2001, pp. 545–554. ISBN: 1581133480. DOI: 10.1145/371920.372156. URL: https://doi.org/10.1145/371920.372156.

[13] Edward G Coffman Jr, Kimming So, Micha Hofri, and AC Yao. "A stochastic model of bin-packing". In: *Information and control* 44.2 (1980), pp. 105–115.

[14] Isabel Correia, Luís Gouveia, and Francisco Saldanha-da-Gama. "Solving the variable size bin packing problem with discretized formulations". In: *Computers Operations Research* 35.6 (2008). Part Special Issue: OR Applications in the Military and in Counter-Terrorism, pp. 2103–2113. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2006.10.014. URL: https://www.sciencedirect.com/science/article/pii/S0305054806002747.

[15] Bhupesh Kumar Dewangan, Amit Agarwal, Tanupriya Choudhury, Ashutosh Pasricha, and Suresh Chandra Satapathy. "Extensive review of cloud resource management techniques in industry 4.0: Issue and challenges". In: *Software: Practice and Experience* 51.12 (2021), pp. 2373–2392.

[16] Marco Dorigo and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem". In: *IEEE Transactions on evolutionary computation* 1.1 (1997), pp. 53–66.

[17] Kushal Dutta, Ramendu Bikash Guin, Sayan Chakrabarti, Sourav Banerjee, and Utpal Biswas. "A smart job scheduling system for cloud computing service providers and users: Modeling and simulation". In: *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*. 2012, pp. 346–351. DOI: 10.1109/RAIT.2012.6194444.

[18] S. El Kafhali and K Salah. "Modeling and Analysis of Performance and Energy Consumption in Cloud Data Centers". In: *Arab J Sci Eng*. Vol. 43. 2018, pp. 7789–7802. DOI: 10.1007/s13369-018-3196-0. URL: https://doi.org/10.1007/s13369-018-3196-0.

[19] A.K. Erlang. "Solution of Some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges". In: *Post Office Electrical Engineer's Journal* 10 (1917), pp. 189–197.

[20] D. K. Friesen and M. A. Langston. "Variable Sized Bin Packing". In: *SIAM Journal on Computing* 15.1 (1986), pp. 222–230. DOI: 10.1137/0215016. URL: https://doi.org/10.1137/0215016.

[21] Noah Gans and Yong-Pin Zhou. "Managing learning and turnover in employee staffing". In: *Operations Research* 50.6 (2002), pp. 991–1006.

[22] Natarajan Gautam. *Analysis of Queues: Methods and Applications*. 1st ed. CRC Press: Springer Berlin Heidelberg, 2012. DOI: 10.1201/b11858. URL: https://doi.org/10.1201/b11858.

[23] Stéphane Grandcolas and Cédric Pinto. "A SAT Encoding for Multi-dimensional Packing Problems". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Ed. by Andrea Lodi, Michela Milano, and Paolo Toth. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 141–146. ISBN: 978-3-642-13520-0.

[24] Tom Guérout, Yacine Gaoua, Christian Artigues, Georges Da Costa, Pierre Lopez, and Thierry Monteil. "Mixed integer linear programming for quality of service optimization in Clouds". In: *Future Generation Computer Systems* 71 (2017), pp. 1–17.

[25] M-A Guerry and T De Feyter. "Optimal recruitment strategies in a multi-level manpower planning model". In: *Journal of the Operational Research Society* 63.7 (2012), pp. 931–940. DOI: 10.1057/jors.2011.99. URL: https://doi.org/10.1057/jors.2011.99.

[26] Lizheng Guo, Tao Yan, Shuguang Zhao, and Changyuan Jiang. "Dynamic performance optimization for cloud computing using M/M/m queueing system". In: *Journal of applied mathematics* 2014 ().

[27] Cengiz Haksever and John Moussourakis. "A model for optimizing multi-product inventory systems with multiple constraints". In: *International Journal of Production Economics* 97.1 (2005), pp. 18–30.

[28] Mohamed Hanini, Said El Kafhali, and Khaled Salah. "Dynamic VM allocation and traffic control to manage QoS and energy consumption in cloud computing environment". In: *International Journal of Computer Applications in Technology* 60.4 (2019), pp. 307–316.

[29] D.P. Heyman and M.J. Sobel. *Stochastic Models in Operations Research: Stochastic Processes and Operating Characteristics*. Dover Books on Computer Science Series. Dover Publications, 2004. ISBN: 9780486432595. URL: https://books.google.ca/books?id=IcVlwPS0qCwC.

[30] O.J. Ibarra-Rojas, F. Delgado, R. Giesen, and J.C. Muñoz. "Planning, operation, and control of bus transport systems: A literature review". In: *Transportation Research Part B: Methodological* 77 (2015), pp. 38–75. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2015.03.002. URL: https://www.sciencedirect.com/science/article/pii/S0191261515000454.

[31] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. "Applying queue theory for modeling of cloud computing: A systematic review". In: *Concurrency and Computation: Practice and Experience* 31.17 (2019), e5186.

[32] Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey. *50 Years of Integer Programming 1958–2008*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-540-68279-0. URL: https://doi.org/10.1007/978-3-540-68279-0.

[33] David G. Kendall. "Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain". In: *The Annals of Mathematical Statistics* 24.3 (1953), pp. 338–354. DOI: 10.1214/aoms/1177728975. URL: https://doi.org/10.1214/aoms/1177728975.

[34] Leonid Khachiyan. "Polynomial algorithms in linear programming". In: *Ussr Computational Mathematics and Mathematical Physics* 20 (1980), pp. 53–72.

[35] Hamzeh Khazaei, Jelena Misic, and Vojislave B Misic. "Modelling of cloud computing centers using M/G/m queues". In: *2011 31st International Conference on Distributed Computing Systems Workshops*. IEEE. 2011, pp. 87–92.

[36] Ger Koole and Avishai Mandelbaum. "Queueing Models of Call Centers: An Introduction". In: *Annals of Operations Research* 113 (July 2002), pp. 41–59. DOI: 10.1023/A:1020949626017.

[37] AH Land and AG Doig. "An Automatic Method of Solving Discrete Programming Problems". In: *Econometrica: Journal of the Econometric Society* (1960), pp. 497–520.

[38] C. Li and L.Y. Li. "Optimal resource provisioning for cloud computing environment". In: *The Journal of Supercomputing* 62 (2012), pp. 989–1022. DOI: https://doi.org/10.1007/s11227-012-0775-9.

[39] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. "Online dynamic capacity provisioning in data centers". In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2011, pp. 1159–1163. DOI: 10.1109/Allerton.2011.6120298.

[40] Martin Shubik Lode Li and Matthew J. Sobel. "Control of Dividends, Capital Subscriptions, and Physical Inventories". In: *Management Science* 59.5 (2013), pp. 1107–1124. DOI: https://doi.org/10.1287/mnsc.1120.1629.

[41] M Donald MacLaren. "The Art of Computer Programming—Volume 1: Fundamental Algorithms (Donald E. Knuth)". In: *Siam Review* 11.1 (1969), pp. 89–91.

[42] Syed Hamid Hussain Madni, Muhammad Shafie Abd Latiff, Yahaya Coulibaly, and Shafi'i Muhammad Abdulhamid. "Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges and opportunities". In: *Journal of Network and Computer Applications* 68 (2016), pp. 173–200. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.04.016. URL: https://www.sciencedirect.com/science/article/pii/S1084804516300674.

[43] Jiří Matoušek and Bernd Gärtner. "Integer Programming and LP Relaxation". In: *Understanding and Using Linear Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 29–40. ISBN: 978-3-540-30717-4. DOI: 10.1007/978-3-540-30717-4_3. URL: https://doi.org/10.1007/978-3-540-30717-4_3.

[44] Per-Olov Östberg, James Byrne, Paolo Casari, Philip Eardley, Antonio Fernandez Anta, Johan Forsman, John Kennedy, Thang Le Duc, Manuel Noya Mariño, Radhika Loomba, Miguel Ángel López Peña, José Lápez Veiga, Theo Lynn, Vincenzo Mancuso, Sergej Svorobej, Anders Torneus, Stefan Wesner, Peter Willis, and Jörg Domaschka. "Reliable capacity provisioning for distributed cloud/edge/fog computing applications". In: *2017 European Conference on Networks and Communications (EuCNC)*. 2017, pp. 1–6. DOI: 10.1109/EuCNC.2017.7980667.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[46] Jamol Pender and Young Myoung Ko. "Approximations for the queue length distributions of time-varying many-server queues". In: *INFORMS Journal on Computing* 29.4 (2017), pp. 688–704. DOI: 10.1287/ijoc.2017.0760.

[47] David Pisinger and Mikkel Sigurd. "Using Decomposition Techniques and Constraint Programming for Solving the Two-Dimensional Bin-Packing Problem". In: *INFORMS Journal on Computing* 19.1 (2007), pp. 36–51. DOI: 10.1287/ijoc.1060.0181.

[48] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[49] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting". In: *2011 IEEE 4th International Conference on Cloud Computing*. 2011, pp. 500–507. DOI: 10.1109/CLOUD.2011.42.

[50] Yatendra Sahu, R.K. Pateriya, and Rajeev Kumar Gupta. "Cloud Server Optimization with Load Balancing and Green Computing Techniques Using Dynamic Compare and Balance Algorithm". In: *2013 5th International Conference and Computational Intelligence and Communication Networks*. 2013, pp. 527–531. DOI: 10.1109/CICN.2013.114.

[51] Upendra Sharma, Prashant Shenoy, Sambit Sahu, and Anees Shaikh. "A Cost-Aware Elasticity Provisioning System for the Cloud". In: *2011 31st International Conference on Distributed Computing Systems*. 2011, pp. 559–570. DOI: 10.1109/ICDCS.2011.59.

[52] Leyuan Shi. "Approximate analysis for queueing networks with finite capacity and customer loss". In: *European Journal of Operational Research* 85.1 (1995), pp. 178–191. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(93)E0252-S. URL: https://www.sciencedirect.com/science/article/pii/0377221793E0252S.

[53] Edmundo de Souza e Silva and Mario Gerla. "Load Balancing in Distributed Systems with Multiple Classes and Site Constraints". In: *Proceedings of the Tenth International Symposium on Computer Performance Modelling, Measurement and Evaluation*. Performance '84. NLD: North-Holland Publishing Co., 1984, pp. 17–33. ISBN: 0444876804.

[54] Sukhpal Singh and Inderveer Chana. "Cloud resource provisioning: survey, status and future research directions". In: *Knowledge and Information Systems* 49.3 (2016), pp. 1005–1069.

[55] Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, and Hidetomo Nabeshima. "A SAT-based method for solving the two-dimensional strip packing problem". In: *Fundamenta Informaticae* 102.3-4 (2010), pp. 467–487.

[56] Biao Song, Mohammad Mehedi Hassan, Atif Alamri, Abdulhameed Alelaiwi, Yuan Tian, Mukaddim Pathan, and Ahmad Almogren. "A two-stage approach for task and resource management in multimedia cloud environment". In: *Computing* 98.1 (2016), pp. 119–145.

[57] Benjamin Speitkamp and Martin Bichler. "A mathematical programming approach for server consolidation problems in virtualized data centers". In: *IEEE Transactions on services computing* 3.4 (2010), pp. 266–278.

[58] János Sztrik et al. "Basic queueing theory". In: *University of Debrecen, Faculty of Informatics* 193 (2012), pp. 60–67.

[59] Muhammad Tirmazi, Adam Barker, Nan Deng, Md Ehtesam Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. "Borg: the Next Generation". In: *EuroSys'20*. Heraklion, Crete, 2020.

[60] Tony T Tran, Meghana Padmanabhan, Peter Yun Zhang, Heyse Li, Douglas G Down, and J Christopher Beck. "Multi-stage resource-aware scheduling for data centers with heterogeneous servers". In: *Journal of Scheduling* 21.2 (2018), pp. 251–267.

[61] Bhat U.N. *An Introduction to Queueing Theory*. 2015.

[62] Hien Nguyen Van, Frédéric Dang Tran, and Jean-Marc Menaud. "Performance and Power Management for Cloud Infrastructures". In: *2010 IEEE 3rd International Conference on Cloud Computing*. 2010, pp. 329–336. DOI: 10.1109/CLOUD.2010.25.

[63] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. "Large-scale cluster management at Google with Borg". In: *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France, 2015.

[64] Jordi Vilaplana, Francesc Solsona, Ivan Teixidó, Jordi Mateo Fornes, Francesc Abella, and J. Rius. "A queuing theory model for cloud computing". In: *The Journal of Supercomputing* 69 (July 2014), pp. 1–16. DOI: 10.1007/s11227-014-1177-y.

[65] William Voorsluys, James Broberg, Rajkumar Buyya, et al. "Introduction to cloud computing". In: *Cloud computing: Principles and paradigms* (2011), pp. 1–44.

[66] Shuang Wang, Xiaoping Li, Quan Z Sheng, Ruben Ruiz, Jinquan Zhang, and Amin Beheshti. "Multi-queue request scheduling for profit maximization in IaaS clouds". In: *IEEE Transactions on Parallel and Distributed Systems* 32.11 (2021), pp. 2838–2851.

[67] Ward Whitt. "APPROXIMATIONS FOR THE GI/G/m QUEUE". In: *Production and Operations Management* 2.2 (1993), pp. 114–161. DOI: https://doi.org/10.1111/j.1937-5956.1993.tb00094.x.

[68] John Wilkes. "Cluster Data". In: *GitHub repository* (2020). URL: https://github.com/google/cluster-data/blob/master/ClusterData2019.md.

[69] Bo Yang, Feng Tan, Yuan-Shun Dai, and Suchang Guo. "Performance Evaluation of Cloud Service Considering Fault Recovery". In: *Cloud Computing*. Ed. by Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 571–576.

[70] Jiangtao Zhang, Hejiao Huang, and Xuan Wang. "Resource provision algorithms in cloud computing: A survey". In: *Journal of Network and Computer Applications* 64 (2016), pp. 23–42. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2015.12.018. URL: https://www.sciencedirect.com/science/article/pii/S1084804516000643.

[71] Long Zhang, Yi Zhuang, and Wei Zhu. "Constraint programming based virtual cloud resources allocation model". In: *International Journal of Hybrid Information Technology* 6.6 (2013), pp. 333–344.

[72] Neng-Fa Zhou, Masato Tsuru, and Eitaku Nobuyama. "A Comparison of CP, IP, and SAT Solvers through a Common Interface". In: *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*. Vol. 1. 2012, pp. 41–48. DOI: 10.1109/ICTAI.2012.15.

[73] Horst Zisgen. "An approximation of general multi-server queues with bulk arrivals and batch service". In: *Operations Research Letters* 50.1 (2022), pp. 57–63. ISSN: 0167-6377. DOI: https://doi.org/10.1016/j.orl.2021.12.006. URL: https://www.sciencedirect.com/science/article/pii/S0167637721001760.