Dynamic Service Placement in Mobile Micro-Clouds

Shiqiang Wang

A Thesis Submitted in Fulfilment of Requirements for the Degree of

Doctor of Philosophy of Imperial College London and

Diploma of Imperial College

Communications and Signal Processing Group

Department of Electrical and Electronic Engineering

Imperial College London

2015

Declaration of Originality

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Shiqiang Wang

Abstract

Cloud computing is an important enabling technique for running complicated applications on resource-limited handheld devices, personal computers, or small enterprise servers, by offloading part of the computation and storage to the cloud. However, traditional centralized cloud architectures are incapable of coping with many emerging applications that are delay-sensitive and require large amount of data exchange between the front-end and back-end components of the application. To tackle these issues, the concept of mobile micro-cloud (MMC) has recently emerged. An MMC is typically connected directly to a network component, such as a wireless basestation, at the edge of the network and provides services to a small group of users. In this way, the communication distances between users and the cloud(s) hosting their services are reduced, and thus users can have more instantaneous access to cloud services.

Several new challenges arise in the MMC context, which are mainly caused by the limited coverage area of basestations and the dynamic nature of mobile users, network background traffic, etc. Among these challenges, one important problem is where (on which cloud) to place the services (or, equivalently, to execute the service applications) to cope with the user demands and network dynamics. We focus on this problem in this thesis, and consider both the initial placement and subsequent migration of services, where migration may occur when the user location or network conditions change.

The problem is investigated from a theoretical angle with practical considerations. We first abstract the service application and the physical cloud system as graphs, and propose online approximation algorithms for finding the placement of

an incoming stream of application graphs onto the physical graph. Then, we consider the dynamic service migration problem, which we model as a Markov decision process (MDP). The state space of the MDP is large, making it difficult to solve in real-time. Therefore, we propose simplified solution approaches as well as approximation methods to make the problem tractable. Afterwards, we consider more general non-Markovian scenarios but assume that we can predict the future costs with a known accuracy. We propose a method of dynamically placing each service instance upon its arrival and a way of finding the optimal look-ahead window size for cost prediction. The results are verified using simulations driven by both synthetic and real-world data traces. Finally, a framework for emulating MMCs in a more practical setting is proposed. In our view, the proposed solutions can enrich the fundamental understanding of the service placement problem. It can also path the way for practical deployment of MMCs. Furthermore, various solution approaches proposed in this thesis can be applicable or generalized for solving a larger set of problems beyond the context of MMC.

Acknowledgments

Looking back to the years of my Ph.D. study, there are many colleagues and friends whom I would like to thank. First, I would like to deeply thank my Ph.D. supervisor Prof. Kin K. Leung, who has been continuously supporting and guiding me throughout these years. Prof. Leung sets an excellent example as a world-class researcher. From him, I have learned not only how to do research, but also how to behave in the scientific community, which I believe will be my standard throughout the rest of my career. Furthermore, he has provided me with great opportunities of working with other outstanding researchers in different parts of the world – a very valuable experience for my professional development.

I sincerely thank my close collaborators, Dr. Rahul Urgaonkar and Dr. Ting He, at IBM T.J. Watson Research Center in New York, Dr. Murtaza Zafer at Nyansa Inc. in California, and Dr. Kevin Chan at the U.S. Army Research Laboratory (ARL) in Maryland. Their superb technical skills and dedication to research greatly impressed me and helped me carry out the research work in this thesis. I have always been enjoying the fruitful collaborations with them. I would also like to thank Dr. Seraphin Calo (IBM) and Dr. Kang-Won Lee (formerly at IBM) who provided me with precious internship opportunities in the summers of 2013 and 2014. I also thank many other collaborators on the U.S./U.K. International Technology Alliance (ITA) Project, including Dr. Theodoros Salonidis (IBM), Dr. Bong-Jun Ko (IBM), Dr. Raghu Ganti (IBM) and Katy Warr (Roke Manor Research), as well as other collaborators of mine. Discussions with them have always been enlightening and bringing new ideas.

With thanks, I would like to acknowledge that my Ph.D. work was funded in

part by the ITA Project and a Scholarship from the Department of Electrical and Electronic Engineering at Imperial College London.

Sincere thanks to my viva examiners Dr. Moez Draief (Imperial College London) and Dr. Ananthram Swami (ARL) for their time in attending my viva and providing insightful feedback.

I finally thank my family and friends, for their endless support and encouragement, which makes me feel happy wherever I am and whatever I encounter.

Shiqiang Wang

Related Publications

The work reported in this thesis has been published/submitted as the following papers.

1. Application Graph Placement (Chapters 2–3)

   (a) **S. Wang**, M. Zafer, and K. K. Leung, "Online workload placement with provable performance guarantees in cloud environments," preprint available, to be submitted for journal publication.

   (b) **S. Wang**, G.-H. Tu, R. Ganti, T. He, K. K. Leung, H. Tripp, K. Warr, and M. Zafer, "Mobile micro-cloud: application classification, mapping, and deployment," in *Proc. of Annual Fall Meeting of ITA 2013*, Oct. 2013.

   (c) **S. Wang**, M. Zafer, K. K. Leung, and T. He, "Security-aware application placement in a mobile micro-cloud," in *Proc. of Annual Fall Meeting of ITA 2013*, Oct. 2013.

2. MDP-Based Approach to Dynamic Service Migration (Chapter 4)

   (a) **S. Wang**, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds based on Markov decision processes," submitted to *IEEE Transactions on Mobile Computing*.

   (b) **S. Wang**, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. of IFIP Networking 2015*, May 2015.

(c) **S. Wang**, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds," in *Proc. of IEEE Military Communications Conference (MILCOM) 2014*, Oct. 2014.

(d) **S. Wang**, R. Urgaonkar, M. Zafer, K. Chan, T. He, and K. K. Leung, "Mobility-driven service migration in mobile micro-clouds," in *Proc. of Annual Fall Meeting of ITA 2014*, Sept. 2014.

3. Dynamic Service Placement with Predicted Future Costs (Chapter 5)

(a) **S. Wang**, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," submitted to *IEEE Transactions on Parallel and Distributed Systems*.

(b) **S. Wang**, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," in *Proc. of Annual Fall Meeting of the ITA 2015*, Sept. 2015.

(c) **S. Wang**, R. Urgaonkar, K. Chan, T. He, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," in *Proc. of IEEE International Conference on Communications (ICC) 2015*, Jun. 2015.

4. Emulation-Based Study (Chapter 6)

(a) **S. Wang**, K. Chan, R. Urgaonkar, T. He, and K. K. Leung, "Emulation-based study of dynamic service placement in mobile micro-clouds," in *Proc. of IEEE Military Communications Conference (MILCOM) 2015*, Oct. 2015.

(b) **S. Wang**, K. Chan, R. Urgaonkar, T. He, and K. K. Leung, "Emulation-based study of dynamic service placement in mobile micro-clouds," in *Proc. of Annual Fall Meeting of the ITA 2015*, Sept. 2015.

The following publications were completed as part of my Ph.D. study, but its contents are not included in this thesis for compactness.

1. Dynamic Service Migration and Workload Scheduling Based on Lyapunov Optimization

   (a) R. Urgaonkar, **S. Wang**, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," in *Proc. of IFIP Performance 2015*, Oct. 2015.

   (b) R. Urgaonkar, **S. Wang**, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in micro-clouds," in *Proc. of Annual Fall Meeting of the ITA 2015*, Sept. 2015.

   (c) R. Urgaonkar, **S. Wang**, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling using Lyapunov optimization," in *Proc. of Annual Fall Meeting of ITA 2014*, Sept. 2014.

2. Other Topics Related to Mobile Micro-Clouds

   (a) **S. Wang**, R. Urgaonkar, T. He, K. Chan, and K. K. Leung, "Distributed workload scheduling with limited control information exchange in mobile micro-clouds," in *Proc. of Annual Fall Meeting of the ITA 2015*, Sept. 2015.

   (b) A. Freeman, K. Warr, H. Tripp, **S. Wang**, K. Leung, "Assessing the applicability of commercial cloud distributed processing techniques to mobile micro-cloud deployments," in *Proc. of Annual Fall Meeting of ITA 2014*, Sept. 2014.

3. Other Topics

   (a) **S. Wang**, L. Le, N. Zahariev, and K. K. Leung, "Centralized rate control mechanism for cellular-based vehicular networks," in *Proc. of IEEE Global Communications Conference (GLOBECOM) 2013*, Dec. 2013.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

1-D	One-Dimensional
2-D	Two-Dimensional
AM	Always Migrate
CORE	Common Open Research Emulator
EMANE	Extendable Mobile Ad-hoc Network Emulator
FLOPs	Floating-Point Operations
IM	Infrequently Migrate
LP	Linear Program
MAH	Moving Average + Hysteresis
MDP	Markov Decision Process
MILP	Mixed-Integer Linear Program
MMC	Mobile Micro-Cloud
OPT	True Optimal Result
vs.	Versus
w.r.t.	With respect to

Mathematical Notations

Below are some generic notations used in this thesis, specific notations will be introduced in each chapter.

$\triangleq$	Is defined to be equal to
$\lvert\mathcal{V}\rvert$	Number of elements in set $\mathcal{V}$
$\mathcal{R} = (\mathcal{V}, \mathcal{E})$	Graph $\mathcal{R}$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$
$e = (v_1, v_2)$	Edge e connects nodes v_1 and v_2
$v \to n$, $e \to l$	Application node v (or link e) is mapped to physical node n (or link l)
$\lVert\varphi_1 - \varphi_2\rVert$ (or $\lvert\varphi_1 - \varphi_2\rvert$ for 1-D cases)	Distance between locations φ_1 and φ_2
$\mathbb{E}\{\cdot\}$	Expected value
P_{ij} or $P_{i,j}$	Transition probability from state i to state j
$\Pr\{X\}$	Probability of random event X
$\mathbf{a} \cdot \mathbf{b}$	Dot-product of vectors $\mathbf{a}$ and $\mathbf{b}$
$(\mathbf{g})_{m_1 m_2}$ (or $(\mathbf{g})_{m_1 m_2 m_3}$)	The (m_1, m_2)th (or (m_1, m_2, m_3)th) element in vector or matrix $\mathbf{g}$
$\frac{du_{n,t}}{dy}(a)$	Derivative of $u_{n,t}(y)$ w.r.t. y evaluated at $y = a$
$\frac{\partial w_{nh,t}}{\partial z_{nh}}(a, b, c)$	Partial derivative of $w_{nh,t}(y_n, y_h, z_{nh})$ w.r.t. z_{nh} evaluated at $y_n = a, y_h = b, z_{nh} = c$
$(\mathbf{y}, \mathbf{z})$	Vector that concatenates vectors $\mathbf{y}$ and $\mathbf{z}$
$\nabla_{\mathbf{x}}$ (or $\nabla_{\mathbf{y},\mathbf{z}}$)	Gradient w.r.t. *each element* in vector $\mathbf{x}$ (or $(\mathbf{y}, \mathbf{z})$)

CHAPTER 1

Introduction

1.1 Overview

Mobile applications have become increasingly popular over recent years, with examples including data streaming, real-time video processing, etc. These applications generally require high data processing capability. However, portable devices (e.g. smartphones) are limited by their size and battery life, which makes them incapable of performing complicated computational tasks. A solution to this problem is to utilize cloud computing techniques [1, 2], where the cloud provides additional data processing and computational capabilities. Such cloud-based applications usually consist of a front-end component running on the mobile device and one or multiple back-end components running on the cloud [3, 4]. This architecture makes it possible to instantiate complicated applications from handheld devices that have limited processing power.

Traditionally, cloud services are provided by centralized data-centers that may be located far away from end-users, which can be inefficient because the user may suffer from long latency and poor connectivity due to long-distance communication [5]. The question on how to provide readily accessible cloud services has been an ongoing challenge, and it has become particularly important in recent years as delay-sensitive and bandwidth-consuming applications emerge. As a result, the concept of *mobile micro-cloud (MMC)* has recently emerged. The core idea of MMC is to move computation closer to users, where small servers or data-centers that can

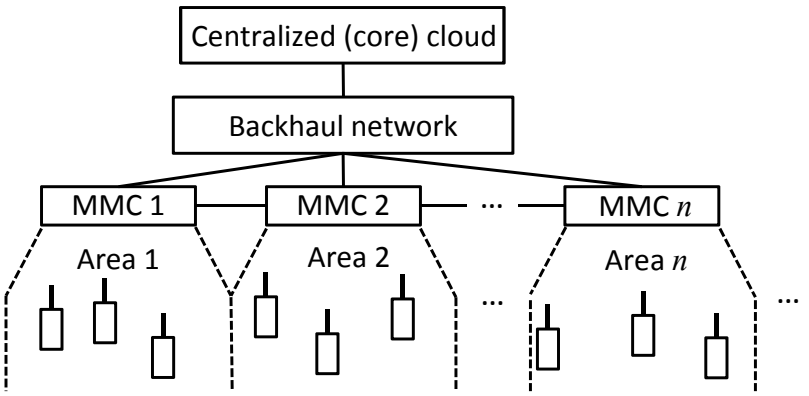
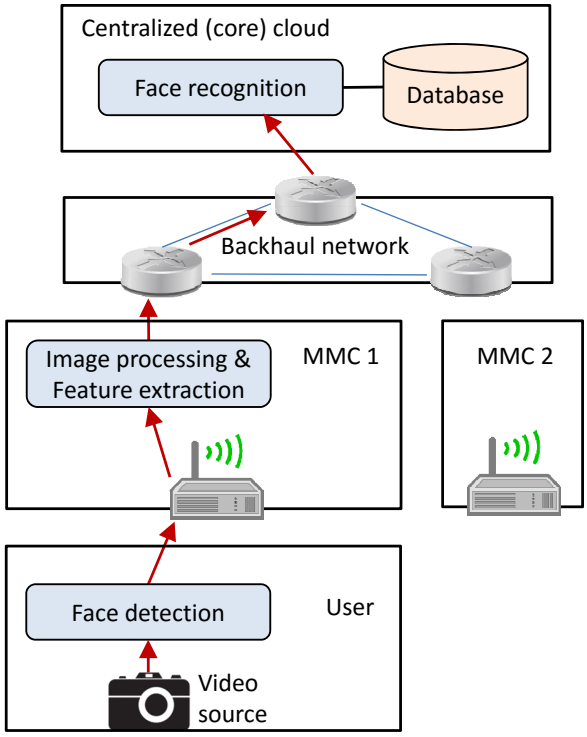


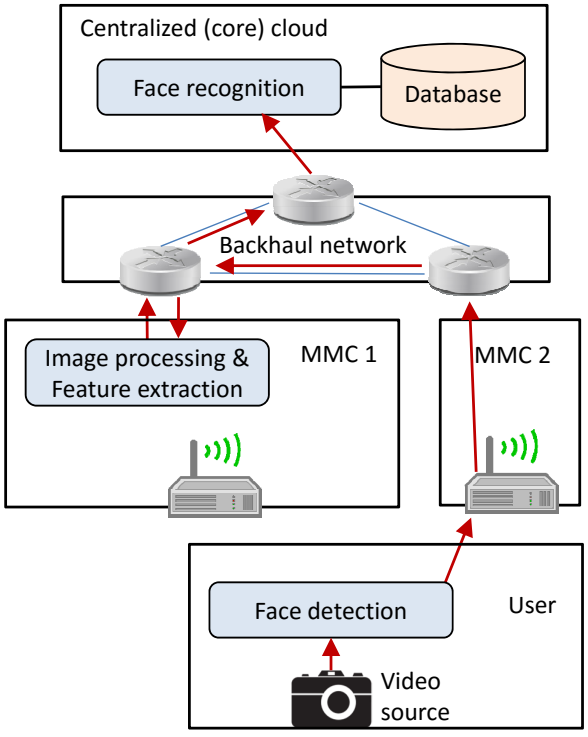
Figure 1.1: Application scenario with mobile micro-clouds (MMCs).

host cloud applications are distributed across the network and connected directly to entities (such as cellular basestations) at the network edge [6, 7]. Fig. 1.1 shows an application scenario where MMCs coexist with the centralized core cloud. The idea of MMCs is also known as cloudlets [8], edge computing [9], fog computing [10], small cell cloud [11], follow me cloud [12], etc. We use the term "MMC" in this thesis. MMCs can be used for many applications that require low latency, high data processing capability, or high reliability [4, 5, 13, 14, 15]. They are expected to develop rapidly with the growth of new mobile applications and more advanced smartphones, and are also more robust than traditional cloud computing systems [8].

One important problem in MMCs is to decide where the computation for each user should be performed, with the presence of user mobility and other dynamic changes in the network. When a user requests for a service provided by the cloud, the service can run either in the centralized cloud or in one of the MMCs. There can also be multiple servers or datacenters within the centralized cloud or within each MMC. The question is how to choose the optimal location to run the service. In addition, the user may move across different geographical areas, thus another question is whether and where should we migrate (move) the service when the user location or network condition changes. We refer to the above problems as the *dynamic service placement problem*, which is the focus of this thesis.



(a)



(b)

Figure 1.2: Example scenario with face recognition application, where the red arrows show the data transmission path: (a) user connected to MMC 1, (b) user connected to MMC 2.

We illustrate the service placement problem with an example face recognition application. This application recognizes faces from a real-time video stream captured by the camera of a hand-held device, as shown in Fig. 1.2. Such applications typically consist of at least three modules: face detection; image processing and feature extraction; and face recognition. Although one can further break each module into smaller building blocks, we consider each module as a whole for simplicity. The service placement problem aims to find the location to place each module. As mentioned above, the problem includes two subproblems: *initial service placement* and *real-time service migration.*

1.1.1 Initial Service Placement

The initial service placement refers to the placement of different modules at the time the service is initialized, i.e., when the face recognition application starts. One possible result of initial service placement is shown in Fig. 1.2(a).

Here, the face detection module is placed at the user, meaning that this module runs directly on the user device. The purpose of face detection is to find areas of an image that contain faces. This task requires relatively low processing capability, and once finished, the application only needs to send specific parts of the image that contain faces for next stage processing. Therefore, it can be beneficial to place this module on the user device, because it saves the necessary communication bandwidth between user and cloud, and at the same time does not consume much processing power of the user device.

The image processing and feature extraction stage usually requires more processing than face detection. Hence, it can be placed on the MMC closest to the user, in order not to overload the user device's processing resource. After processing at this stage, only the extracted features need to be sent to the next stage (face recognition, which is placed on the centralized cloud), thus such a placement reduces the amount

of data sent through the backhaul network.

The reason for placing the face recognition module on the centralized cloud is that this module often needs to frequently look up a large database, which contains features of different people's faces. It is often impractical to transfer the whole database to MMCs due to its size. Thus this placement appears to be good from our reasoning.

Most realistic applications are more complex than our example application here. Therefore, a conceptual analysis such as the above is not always possible. We need to develop algorithms for finding optimal placement. Multiple services may also arrive to the cloud system over time, thus we need to consider such online service arrivals when developing placement algorithms.

1.1.2 Real-Time Service Migration

After the initial service placement, the mobile user may move to a different location. One possibility after such a movement is that the user is now associated to a different MMC, and it has to send data via the backhaul network if the related application module is still running in the previous MMC, as shown in Fig. 1.2(b). Obviously, in Fig. 1.2(b), the communication overhead can be reduced if we migrate the image processing and feature extraction module from MMC 1 to MMC 2. However, as part of migration, data related to the state of the application module usually needs to be transmitted from the original MMC 1 to the new MMC 2. Migration decisions need to be made in a considerate way to achieve a good tradeoff between the migration cost and benefits after migration. A well-designed algorithm is usually required for making such decisions.

Except for user mobility, other factors that may trigger migration include change in resource availability (caused by factors other than user mobility) and mobility of MMCs (for scenarios where MMCs are installed on moving vehicles, for example).

We mainly focus on user mobility as a dominating cause for migration in this thesis, but our results can be extended to more general cases by incorporating parameters in the cost functions to reflect resource availability and considering the relative mobility of users with respect to MMCs.

1.2 Motivation

The previous section has shown the importance of the service placement problem. This section outlines the main gaps between existing literature and what we would like to achieve in this thesis. Detailed literature review on specific aspects are included in each chapter later on.

Cloud computing is a form of distributed computing, where computation is distributed across multiple machines and data exchange is necessary over the communication network. Such a distributed system has many benefits. For example, as mentioned before, resource intensive processing tasks can be offloaded to more powerful computers; processing components that require access to large databases can be placed close to the database, to reduce the network communication overhead. In the traditional setting, cloud services are provided by large datacenters, in which the allocation of jobs to servers (i.e., service placement) and communication within the datacenter are generally predictable and can usually be scheduled by a centralized scheduler.

The idea of MMC was introduced only a few years ago, driven by the increasing popularity and demand of highly reliable and low-latency cloud applications, such as the face recognition application mentioned in Section 1.1. Compared to traditional centralized clouds, MMCs have the following features:

1. Computation is distributed across a large geographical area, whereas in the centralized cloud, computation is usually only distributed within the datacen-

ter.

2. There is a hierarchy from the core centralized cloud down to the MMCs at the network edge. When we allow multiple layers of MMCs, those MMCs that are closer to the network edge serve a smaller area.

3. Compared to centralized clouds, MMCs exhibit much more dynamics in resource availability, due to user mobility and other uncontrollable factors in the network and the cloud (see Section 1.1.2).

The above features cause many existing approaches for service placement in centralized cloud computing [16, 17] inapplicable to MMC, because those approaches do not consider the presence of dynamically changing resource availability related to user mobility. This is a fundamental challenge in MMCs. Conversely, the hierarchical nature of MMCs can enable more efficient service placement algorithms.

We also note that most existing literatures on MMC focus on system aspects [4, 5, 8, 9, 10, 12, 13, 14, 15], while only a few consider its theoretical aspects under simplified settings [11, 18, 19]. The goal of our research is therefore to enrich the theoretical understanding of MMCs, with focus on the dynamic service placement problem. Our results include a set of efficient placement algorithms with provable performance guarantees, which provide insights for practical deployment and are also directly applicable in practice.

1.3 Summary of Contributions

As illustrated by the example in Section 1.1, the service placement problem contains two subproblems. One is the initial placement of services, and the other is the real-time service migration due to dynamic changes. We consider both aspects in this thesis.

The main contributions of this thesis are summarized as follows:

(a) We first model the resource demand of a service with an application/service graph and the resource availability of the physical computing system with a physical graph. The resource demand/availability are annotated on each node and link of these graphs, and there can be multiple types of computational resources at nodes. The service placement problem is then essentially the problem of placing the application graph onto the physical graph, where the node and link assignment[1] are jointly considered. With this model, we formulate the offline placement problem for general graphs as a mixed-integer linear program (MILP) in Section 2.2, with consideration of practical domain and conflict constraints related to security and access-control policies.

(b) After proving that the general graph placement problem is NP-hard even in the offline case, we consider in Section 3.3 the placement of a linear application graph onto a tree physical graph and propose an algorithm for finding its optimal solution.

(c) Based on the line-to-tree placement algorithm proposed in (b), in Section 3.4, we generalize the formulation and propose online approximation algorithms with poly-log competitive ratio[2] for the placement of multiple tree application graphs that arrive over time.

(d) After the initial service placement in Chapters 2 and 3, service migration may be needed due to user and network dynamics. Thus, we consider the dynamic service migration problem under real-time variations in the user location and

[1] We exchangeably use the terms "placement", "assignment", "mapping", and "configuration" in this thesis.

[2] See Section 2.3 for definition of competitive ratio. The term "poly-log" means "polynomial logarithmic". For example, if an algorithm's competitive ratio is $\log^2(N)$, where N is the problem size, we can say that the competitive ratio is poly-log in N.

resource availability, where we first model the problem as a Markov decision process (MDP) in Section 4.2.

(e) We show in Section 4.3 that under one-dimensional (1-D) user mobility and constant cost values, the optimal policy of the MDP (defined in (d)) is a threshold policy. An algorithm is proposed to find this optimal threshold policy, which is more efficient than standard MDP solution approaches.

(f) To incorporate more general cases, we consider two-dimensional (2-D) user mobility with a constant-plus-exponential cost model in Section 4.4. We approximate the state space of the MDP (defined in (d)) by the distance between the user and service locations. We show that the resulting MDP is exact for uniform 1-D random walk user mobility while it provides a close approximation for uniform 2-D random walk mobility with a constant additive error term. A new algorithm and a numerical technique is proposed for computing the optimal policy, which is significantly faster than traditional approaches. We also discuss how to apply the proposed algorithm in real-world scenarios where many theoretical assumptions are relaxed.

(g) For more general cases, the MDP-based approach can be inapplicable. However, if there is an underlying mechanism to predict the future costs, we can find an approximately optimal service placement sequence that minimizes the average cost over time. In Chapter 5, we first propose a method which solves for the optimal placement sequence for a specific look-ahead time window, based on the predicted costs in this time window. We show that when there exist multiple services, the problem is NP-hard, and propose an online approximation algorithm with provable performance guarantee to find the solution. Then, we propose a method to find the optimal look-ahead window size based on the prediction error, which minimizes an upper bound of the average cost.

(h) The above study is theoretical in nature where realistic aspects such as delay in control message exchange are not considered. To study the performance of dynamic service placement in a more practical setting, we propose an emulation framework in Chapter 6, in which physical nodes are encapsulated into virtual containers that are connected via emulated physical links. A real computer program with some basic packet exchange functionalities runs in each node in the emulation. Emulation results of different service placement policies are shown and their insights are discussed.

1.4 Organization of the Thesis

This thesis is organized as follows. Chapter 2 introduces the general problem and related techniques that are used in this thesis. Chapter 3 presents the optimal algorithm for line-to-tree placement and online approximation algorithms for the placement of an incoming stream of tree application graphs. The MDP-based approach for dynamic service migration is presented in Chapter 4. Chapter 5 presents algorithms for dynamic service placement with predicted future costs. An emulation framework is presented in Chapter 6. Chapter 7 draws conclusions and discusses some future directions.

CHAPTER 2

Background of Dynamic Service Placement

2.1 Definitions

We can abstract the service placement problem as a graph (or, as a special case, node) placement problem, as illustrated in Fig. 2.1. In the following, we introduce some concepts that will be used in this thesis.

Application/Service Graph: A service or a service instance[1] can be abstracted as a graph (referred to as the *application/service graph*[2]) in which the nodes represent processing/computational modules in the service, and the edges represent communication demand between the nodes. Each node $v \in \mathcal{V}$ in the application graph $\mathcal{R} = (\mathcal{V}, \mathcal{E})$ is associated with parameters that represent the computational resource

[1]For simplicity, we exchangeably use the terms "service" and "service instance" in this thesis.
[2]We exchangeably use the notions application graph and service graph in this thesis.

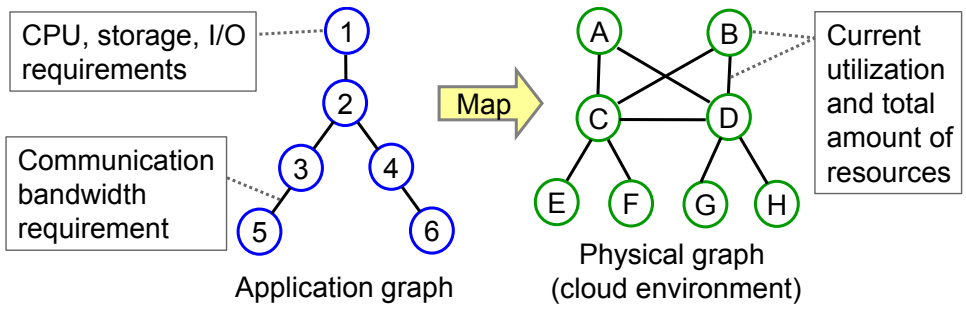


Figure 2.1: The service placement problem.

(of K different types) demands of node v. Similarly, each edge $e \in \mathcal{E}$ is associated with a communication bandwidth demand. The notation $e = (v_1, v_2)$ denotes that application edge e connects application nodes v_1 and v_2. The application graph $\mathcal{R}$ can be either a directed or an undirected graph. If it is a directed graph, the direction of edges specify the direction of data communication; if it is an undirected graph, data communication can occur in either direction along application edges.

Physical Graph: The physical computing system can also be abstracted as a graph (referred to as the *physical graph*) with nodes denoting network elements such as data-centers[3], servers, routers, etc. and edges denoting communication links between the nodes. Each node $n \in \mathcal{N}$ in the physical graph $\mathcal{Y} = (\mathcal{N}, \mathcal{L})$ has K different types of computational resources, and each edge $l \in \mathcal{L}$ has a communication capability. For nodes in the physical network that do not have capability of hosting computational modules (such as routers), we can regard their computational capacity as zero. We use the notation $l = (n_1, n_2)$ to denote that physical link l connects physical nodes n_1 and n_2. Similar to the application graph, the physical graph can be either directed or undirected, depending on whether the physical links are bidirectional (i.e., communication in both directions share the same link) or single-directional (i.e., communication in each direction has a separate link).

Policy: A *policy* specifies how to place the application graph $\mathcal{R}$ onto the physical graph $\mathcal{Y}$. In a dynamic setting, the policy can also depend on the current state of the system (defined below), so that a new placement decision can be made based on the current placement. It can also be dependent on the time of performing the placement.

[3] A physical node can either represent a data-center (which is a collection of network elements) or a single network element, depending on the granularity we consider in practical cases.

State: The *state* of the cloud system specifies where each application node and link is placed, and the current status (including topology, resource demand/availability, etc.) of the application and physical graphs.

Cost: Each possible way of service placement incurs some costs. Generally, there are two types of costs. The first type is the cost of running services on the cloud system when the placement of these services remain unchanged. The second type is the migration cost, which is incurred when we change the placement of the service during its operation. We consider the migration cost because when the service placement is altered, we may need to restart the service or transfer some state information to the new physical machine that runs the service, which incurs some cost. The explicit cost definition will be discussed in details for each specific problem in the remainder of this thesis.

Constraints: There can be constraints such as the resource capacity of each physical node and link, the location that each application node and link can be placed at, as well as other constraints including those that are related to access or security restrictions.

Offline and Online Service Placement: Throughout this thesis, we say that a service placement is *offline* when our goal is to place a single or a set of application graphs "in one shot". In contrast, an *online* service placement is the case where we have an incoming stream of application graphs, which have to be sequentially placed onto the physical graph as each application graph arrives. There may or may not exist some application graphs that depart while others are arriving.

A Note on Graphs: For the problems discussed Chapters 4, 5, and 6, the notion of graphs is not important for explaining the problems. In those chapters, we will

not explicitly use the notion of graphs in the problem formulation, but one can easily define those problems based on application and physical graphs.

2.2 A Mixed-Integer Linear Program (MILP) Approach to Offline Service Placement

To provide a basic understanding to service placement, in this section, we illustrate how an offline service placement problem can be formulated as an MILP, and show an example of the placement result.

The goal of service placement is to place the service onto the physical cloud. With the notion of application and physical graphs, the service placement problem can be seen as a graph mapping problem, which maps each application node to one physical node and each application link to at least one physical path connecting the two application nodes.

A feasible mapping is usually subject to several constraints that need to be satisfied. For example, the resource capacity at a physical node or link can be finite, and the sum resource consumption of application nodes/links that the physical node/link is hosting cannot exceed this limit. In addition, there can be constraints on where the application nodes can be placed and which set of application nodes cannot be hosted on the same physical node. We call these two types of constraints respectively as *domain and conflict constraints* as illustrated in Fig. 2.2. These constraints can be related to security policies, database locations, etc.

All the above-mentioned constraints can be expressed as a set of linear constraints with some binary variables. As a result, the service placement problem can be formulated as an MILP, as described next.

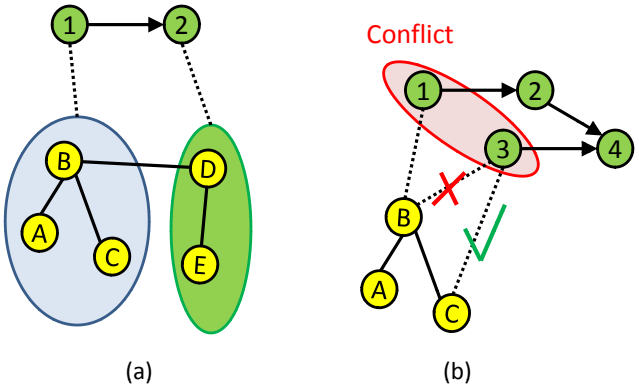


Figure 2.2: Domain and conflict constraints: (a) domain constraint – application node 1 can only be mapped to physical nodes A, B, and C and application node 2 can only be mapped to physical nodes D and E; (b) conflict constraint – application nodes 1 and 3 cannot be mapped onto the same physical node.

2.2.1 Additional Definitions

In the following, we summarize several definitions in addition to those introduced in Section 2.1, which are used in the MILP formulation. We assume that the application graph $\mathcal{R}$ is a directed graph (where the direction of each edge specifies the data flow direction) and the physical graph $\mathcal{Y}$ is an undirected graph. Other cases can also be formulated using MILP but we omit the discussion here for simplicity.

Each node $v \in \mathcal{V}$ is associated with demands $\phi_{v,1}, \phi_{v,2}, ..., \phi_{v,K}$ which represent the computation resource (of K different types) demand of node v. Similarly, each edge $e \in \mathcal{E}$ is associated with a communication bandwidth demand ψ_e. The capacity of type k resource at node $n \in \mathcal{N}$ is $c_{n,k}$, and the communication bandwidth capacity of edge $l \in \mathcal{L}$ is c_l. For nodes in the physical network that do not have capability of hosting service modules (such as routers), we can set $c_{n,k} = 0$ for $\forall k = \{1, 2, ..., K\}$.

Let $\mathcal{N}_m(v) \subseteq \mathcal{N}$ for $\forall v \in \mathcal{V}$ denote the subset of physical nodes that v can be mapped to, and $\mathcal{N}_m^2(v_1, v_2) \subseteq \mathcal{N} \times \mathcal{N}$ for $\forall v_1, v_2 \in \mathcal{V}$ denote the subset of physical node-pairs that the application node-pair (v_1, v_2) can be mapped to. We also use the notation $v \rightarrow n$ to denote that the application node v is mapped to physical node n, and we use $e \rightarrow l$ to denote that the application link e is mapped to physical link l.

Define the normalized demand of type k resource of placing v to n as

$$d_{v\to n,k} = \begin{cases} \frac{\phi_{v,k}}{c_{n,k}}, & \text{if } n \in \mathcal{N}_m(v) \\ \infty, & \text{otherwise} \end{cases}. \tag{2.1}$$

Define the normalized resource demand of placing $e = (v_1, v_2)$ to $l = (n_1, n_2)$ as

$$b_{e\to l} = \begin{cases} \frac{\psi_e}{c_l}, & \text{if } (n_1, n_2) \in \mathcal{N}_m^2(v_1, v_2) \\ \infty, & \text{otherwise} \end{cases}. \tag{2.2}$$

2.2.2 MILP Formulation

We first define the decision variables in our optimization problem. The flow variables $f_{e\to(n_1,n_2)}$ for $\forall e \in \mathcal{E}$ and $\forall n_1, n_2 \in \mathcal{N}$ that have a communication link connecting them denote the amount of data sent from n_1 to n_2, for the application edge e. The binary variables $x_{v\to n}$ for $\forall v \in \mathcal{V}, \forall n \in \mathcal{N}_m(v)$ indicate whether v is mapped to n. Auxiliary variables $y_{v\to n}$ for $\forall v \in \mathcal{V}, \forall n \in \mathcal{N}_m(v)$ indicate whether a node that has a conflict with v is mapped to n, which is used to assist the optimization problem formulation.

We focus on the case where $K = 1$ in our formulation in this section, but it can be easily extended to other values of K.

To jointly consider the load balancing of servers and communication links, as well as minimizing the total amount of communication bandwidth consumption, we

consider the following objective function:

$$\begin{aligned}\min \quad & \Bigg\{ \max_{n\in\mathcal{N}} \alpha_n \left(\sum_{v\in V} d_{v\to n,1} x_{v\to n} \right) \\ & + \max_{l=(n_1,n_2)\in\mathcal{L}} \beta_l \left(\sum_{e\in E} \frac{f_{e\to(n_1,n_2)} + f_{e\to(n_2,n_1)}}{c_l} \right) \\ & + \sum_{l=(n_1,n_2)\in\mathcal{L}} \beta'_l \sum_{e\in E} \frac{f_{e\to(n_1,n_2)} + f_{e\to(n_2,n_1)}}{c_l} \Bigg\}, \end{aligned} \tag{2.3}$$

where the edge l connects n_1 and n_2, and α_n, β_l, and β'_l are weighting factors. This is not the only possible form of objective function. One can define other types of objective functions and still formulate the problem as an MILP as long as the objective function is linear in the decision variables.

The objective function (2.3) can be rewritten as a linear objective function, by adding two additional sets of constraints for the maximum operations, as follows:

$$\begin{aligned}\min \quad & t_1 + t_2 + \sum_{l=(n_1,n_2)\in L} \beta'_l \sum_{e\in E} \frac{f_{e\to(n_1,n_2)} + f_{e\to(n_2,n_1)}}{c_l} \\ s.t. \quad & \alpha_n \sum_{v\in V} d_{v\to n,1} x_{v\to n} \le t_1, \forall n \in \mathcal{N} \\ & \max_{l=(n_1,n_2)\in L} \beta_l \sum_{e\in E} \frac{f_{e\to(n_1,n_2)} + f_{e\to(n_2,n_1)}}{c_l} \le t_2, \forall l = (n_1,n_2) \in \mathcal{L} \\ & \text{Other constraints.} \end{aligned} \tag{2.4}$$

The other constraints are discussed in the following.

The node and edge capacity constraints are:

$$\sum_{v\in\mathcal{V}} d_{v\to n,1} x_{v\to n} \le 1, \forall n \in \mathcal{N}, \tag{2.5}$$

$$\sum_{e\in\mathcal{E}} \left(f_{e\to(n_1,n_2)} + f_{e\to(n_2,n_1)} \right) \le c_l, \forall l = (n_1,n_2) \in \mathcal{L}, \tag{2.6}$$

The flow conservation constraint is:

$$\sum_{n_2:l=(n_1,n_2)\in\mathcal{L}} f_{e\to(n_1,n_2)} - \sum_{n_2:l=(n_1,n_2)\in\mathcal{L}} f_{e\to(n_2,n_1)}$$
$$= c_l b_{e\to l}\left(x_{v_1\to n_1}\mathbf{1}_{n_1\in\mathcal{N}_m(v_1)} - x_{v_2\to n_1}\mathbf{1}_{n_1\in\mathcal{N}_m(v_2)}\right),$$
$$\forall n_1\in\mathcal{N},\forall e\in\mathcal{E}, \tag{2.7}$$

where e is the directed application edge from v_1 to v_2, and the indicator $\mathbf{1}$ indicates whether the condition in its subscript is satisfied. Note that these conditions are static and do not change with the decision variables in the optimization.

The following constraint guarantees that each application node is mapped to exactly one physical node:

$$\sum_{n\in\mathcal{N}_m(v)} x_{v\to n} = 1, \forall v\in\mathcal{V}. \tag{2.8}$$

The node conflict constraints are:

$$y_{v\to n} \le \sum_{(n,n)\notin\mathcal{N}_m^2(v,v_1)} x_{v_1\to n}, \forall v\in\mathcal{V},\forall n\in\mathcal{N}, \tag{2.9}$$

$$y_{v\to n} \ge \frac{\sum_{(n,n)\notin\mathcal{N}_m^2(v,v_1)} x_{v_1\to n}}{|\mathcal{V}|}, \forall v\in\mathcal{V},\forall n\in\mathcal{N}, \tag{2.10}$$

$$x_{v\to n} + y_{v\to n} \le 1, \forall v\in\mathcal{V},\forall n\in\mathcal{N}. \tag{2.11}$$

We also require that $f_{e\to(n_1,n_2)} \ge 0$, $x_{v\to n}\in\{0,1\}$, and $y_{v\to n}\in\{0,1\}$.

2.2.3 Example Mapping Result

Our optimization problem is an MILP, which can be solved with IBM CPLEX [20], or OPTI Toolbox [21], etc. Fig. 2.3 shows an example scenario and its mapping result which has been obtained with OPTI Toolbox. In the example, application

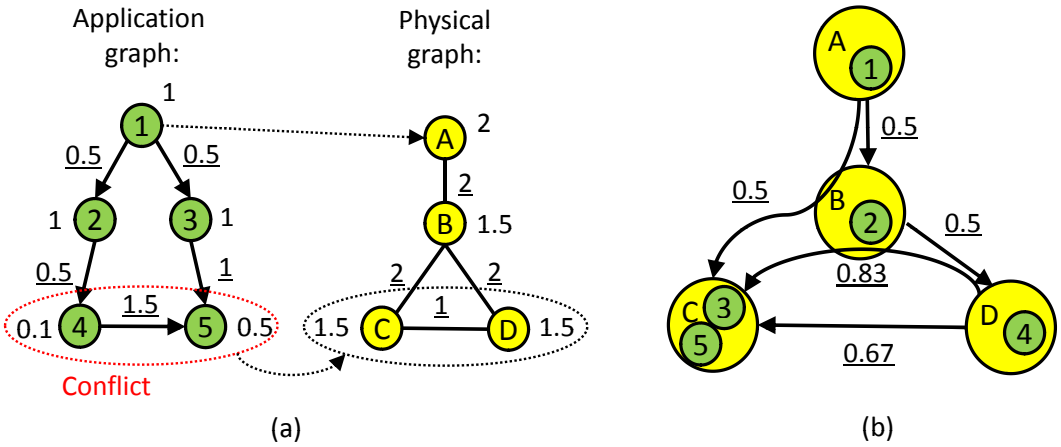


Figure 2.3: Example of service placement: (a) problem setting, (b) mapping result. In (a), the numbers besides nodes and edges in the application graph are resource demands, and the numbers in the physical graph are capacity values, the underlined numbers correspond to edge values. In (b), the underlined numbers are the bandwidth consumption on the corresponding communication links.

node 1 can only be mapped to physical node A, and application nodes 4 and 5 can only be mapped to physical nodes C and D but not on the same node. We set $\alpha_n = \beta_l = \beta'_l = 1$, the demands and capacities are indicated in Fig. 2.3(a) and the mapping result is shown in Fig. 2.3(b).

2.3 Approximation Algorithms

The MILP formulated in the above section is NP-hard. In fact, the service placement problem is NP-hard even for simple graphs as we will discuss later in Section 3.4.1. The term "NP-hard" refers to a particular problem class for which it is generally believed (although a formal proof is still an open question in algorithms research) that the simplest algorithms for solving such problems require exponential time-complexity. If an algorithm has exponential complexity, the complexity of the algorithm scales badly with the size of problem input. For example, in the service placement problem, the input size can be reflected by the number of nodes in both the application and physical graphs. It turns out that for large graphs, an algorithm

with exponential complexity requires a substantial amount of time to find the optimal solution, which can easily become unacceptably time consuming. Therefore, a common practice is to employ heuristic algorithms with polynomial time-complexity to solve such problems [16, 22]. Such algorithms find suboptimal solutions instead of optimal solutions.

For most existing heuristic algorithms, the performance is evaluated via simulation or experimental studies under a limited set of application scenarios. A natural question is: Do such heuristics perform well enough in all the scenarios that possibly occur? Obviously, an empirical study based on simulations or experiments is not sufficient to answer this question. We need to seek theoretical foundations to justify the performance of these heuristic algorithms.

One theoretical basis to study the performance of algorithms that approximate the solution to NP-hard problems is the field of approximation algorithms [23], wherein the worst-case difference between the solution from the approximation algorithm and the true optimal result (OPT) is quantified. We define the *approximation ratio* of a minimization problem as the maximum (worst-case) ratio (either exact or in an upper bound sense) of the result provided by the algorithm to OPT.

Consider the service placement problem introduced earlier in this chapter. For the offline placement problem, we would like to place one or multiple application graphs at a single time. Suppose that we have defined a proper objective (cost) function (such as the one in (2.3)) and would like to *minimize* this cost function. Since this problem is NP-hard, we may want to find an approximately optimal solution to it using algorithms with polynomial time-complexity.

Now, assume that we are given a problem input $\mathcal{I}$. This problem input $\mathcal{I}$ specifies the topology of the application and physical graphs and all parameters associated with them. It also includes any additional input parameters to the problem. Assume we know that the optimal placement incurs a cost equal to $\mathrm{OPT}_{\mathcal{I}}$. If our algorithm

incurs a cost equal to $\mathrm{ALG}_{\mathcal{I}}$, we say that the approximation ratio of this algorithm is Γ_A if

$$\Gamma_A \leq \max_{\mathcal{I}} \frac{\mathrm{ALG}_{\mathcal{I}}}{\mathrm{OPT}_{\mathcal{I}}} \tag{2.12}$$

where the maximization is over *all possible problem inputs*. We say the approximation ratio is *tight* if equality is attained in (2.12).

For the online service placement problem, multiple application graphs arrive over time, where there may or may not exist application graphs leaving while others are arriving. Here, instances (i.e., application graphs in our problem) arrive in real-time and an online algorithm needs to provide a solution at every time when an instance arrives. The competitive ratio is defined as the maximum (worst-case) ratio (either exact or in an upper bound sense) of the result from the *online algorithm* to the *offline* OPT, where the offline OPT is the true optimum by assuming that all the instances are known in advance [24]. We also introduce the notion of an online algorithm being c-*competitive* if its competitive ratio is c.

Let us denote the problem input in the online case as $\mathcal{I}(M)$, where M is the number of application graphs that have arrived to the system, among which some graphs may have already departed from the system. This input contains M instances (application graphs) that arrive one-by-one, and the online algorithm places them onto the physical graph right after an application graph arrives, without prior knowledge on future arrivals. We define $\mathrm{OPT}_{\mathcal{I}(M)}$ as the cost of an optimal *offline* placement, and define $\mathrm{ALG}_{\mathcal{I}(M)}$ as the cost of the placement found by the *online* algorithm. We say that the competitive ratio in placing M instances of this algorithm is Γ_C if

$$\Gamma_C \leq \max_{\mathcal{I}(M)} \frac{\mathrm{ALG}_{\mathcal{I}(M)}}{\mathrm{OPT}_{\mathcal{I}(M)}} \tag{2.13}$$

where the maximization is over all possible problem inputs that have M instances. In some cases, the competitive ratio holds for an arbitrary value of M, and we may

omit the discussion on M in such cases.

The main difference between the competitive and approximation ratios is that the competitive ratio considers the optimality gap with respect to (w.r.t.) an online algorithm, whereas the approximation ratio considers the optimality gap w.r.t. an offline algorithm. The competitive ratio is the same as the approximation ratio if we solve an offline problem in an online way, e.g., breaking the application graph into smaller subgraphs, and assuming online arrival of each of these subgraphs.

Besides using the notions of approximation and competitive ratios, the optimality guarantee can also be quantified in many other ways, such as in the form of an additive error term that specifies the difference between the optimum and its approximate value. We will be considering different forms of optimality guarantees in this thesis.

CHAPTER 3

Online Placement of Application Graphs

3.1 Introduction

Due to the potential hardness of solving the MILP in Section 2.2, we consider approximation algorithms with polynomial time-complexity and provable performance guarantees in this chapter. We start with a basic building block that solves for the placement of a *linear* application graph. We show that for this case, it is actually possible to find the optimal placement (under some constraints) and we propose an algorithm for finding this placement. We then show that for more general cases, for example when the application graph is a tree, the problem is NP-hard. Afterwards, we generalize the algorithm for linear graph placement and obtain online approximation algorithms with poly-log[1] competitive ratio for tree application graph placement. In our formulation, similar to Section 2.2, we jointly consider node and link assignment and incorporate multiple types of computational resources at nodes.

[1]The term "poly-log" means "polynomial logarithmic". For example, if an algorithm's competitive ratio is $\log^2(N)$, where N is the problem size, we can say that the competitive ratio is poly-log in N.

3.1.1 Related Work

In the literature, there is limited work on approximation algorithms with provable approximation/competitive ratios for the service placement problem, especially when it involves both server (node) and network (link) optimization.

In [17], the authors proposed an algorithm for minimizing the sum cost with some considerations on load balancing, which has an approximate approximation ratio of $O(N)$, where N is the number of nodes in the physical graph. The algorithm is based on linear program (LP) relaxation[2], and only allows one node in each application graph to be placed on a particular physical node; thus, excluding server resource sharing among different nodes in one application graph. A drawback of this approach is that the approximation ratio of $O(N)$ is trivial, in the sense that one would achieve the same approximation ratio when placing the whole application graph onto a single physical node instead of spreading it across the whole physical graph.

A theoretical work in [25] proposed an algorithm with $N^{O(D)}$ time-complexity and an approximation ratio of $\delta = O(D^2 \log(ND))$ for placing a tree application graph with D levels of nodes onto a physical graph. It uses LP relaxation and its goal is to minimize the sum cost. Based on this algorithm, the authors showed an online algorithm for minimizing the maximum load on each node and link, which is $O(\delta \log(N))$-competitive when the service durations are equal. The LP formulation in [25] is complex and requires $N^{O(D)}$ variables and constraints. This means when D is not a constant, the space-complexity (specifying the required memory size of the algorithm) is exponential in D.

Another related theoretical work which proposed an LP-based method for offline placement of paths into trees was reported in [26], where the application nodes can

[2]LP relaxation is a common technique used for approximating MILPs. It first replaces the integer variable with a continuous variable, so that the resulting problem becomes an LP which can be solved in polynomial time. Then, a rounding procedure is usually applied to round the values of relaxed integer variables to integers.

only be placed onto the leaves of a tree physical graph, and the goal is to minimize link congestion. In our problem, the application nodes are distributed across users, MMCs, and core cloud, thus they should not be only placed at the leaves of a tree so the problem formulation in [26] is inapplicable to our scenario. Additionally, [26] only focuses on minimizing link congestion. The load balancing of nodes is not considered as part of the objective, whereas only the capacity limits of nodes are considered.

Some other related work focuses on graph partitioning, such as [27] and [28], where the physical graph is defined as a complete graph with edge costs associated with the distance or latency between physical servers. Such an abstraction may combine multiple actual network links into one (abstract) physical edge. Thus, it can be inappropriate when we would like to consider load balancing across all actual network links, which is what we consider in this chapter.

One important aspect is that most existing work, including [17, 26, 27], and [28], do not specifically consider the online operation of the algorithms. Although some of them implicitly claim that one can apply the algorithm repeatedly for each newly arriving service/application, the competitive ratio for such application is not clear. To the best of our knowledge, [25] is the only work that studied the competitive ratio of the online service placement problem that considers both node and link optimization.

3.1.2 Our Approach

In this chapter, we propose algorithms for solving the online service placement problem with provable competitive ratios. Different from [25], our approach is *not* based on LP relaxation. Instead, our algorithms are built upon a baseline algorithm that provides an *optimal* solution to the placement of a linear application graph, i.e., an application graph that is a line. This is the main novelty in contrast to [25], because no optimal solution for linear application graph placement was presented in [25].

Many applications expected to run in an MMC environment can be abstracted as hierarchical graphs, and the simplest case of such a hierarchical graph is a line (see the example in Section 1.1). Therefore, the placement of a linear application graph is an important problem in the context of MMCs.

Another novelty in our work, compared to [25] and most other approaches based on LP relaxation, is that our solution approach is *decomposable* into multiple small building blocks. This makes it easy to extend our proposed algorithms to a distributed solution in the future, which would be very beneficial for reducing the amount of necessary control information exchange among different cloud entities in a distributed cloud environment containing MMCs. This decomposable feature of algorithms also makes it easier to use these algorithms as a sub-procedure for solving a larger problem.

It is also worth noting that the analytical methodology we use in this chapter is new compared to existing techniques such as LP relaxation, thus we enhance the set of tools for online optimization. The theoretical analysis in this chapter also provides insights on the features and difficulty of the problem, which can guide future practical implementations. In addition, the proposed algorithms themselves are relatively easy to implement in practice.

3.1.3 Motivations and Main Results

We propose non-LP based approximation algorithms for online service placement in this chapter, which have both theoretical and practical relevance. The general problem of service placement is hard to approximate even from a theoretical point of view [25, 26, 29]. Therefore, similar to related work [17, 25, 26, 27, 28], we make a few simplifications to make the problem tractable. These simplifications and their motivations are described as follows.

Throughout this chapter, we focus on application and physical graphs that have

tree topologies. This restriction makes the problem mathematically tractable which allows us to obtain exact and approximation algorithms with provable guarantees. Thus, from a theoretical perspective, this work attempts to provide rigor to the service placement problem in a cloud environment.

In terms of practical applicability, a tree application graph models a wide range of practical applications that involve a hierarchical set of processes (or virtual machines). For example, a virtual graph for applications involving streaming, multicasting, or data aggregation is typically represented as a hierarchical set of operators forming a tree topology [30, 31, 32]. Similarly, a web-application topology consisting of a front-end web-server connected to a set of load balancers which are then connected to databases has a tree topology.

For the physical system, the main motivation for us to consider tree physical graphs is the hierarchical nature of MMCs, as discussed in Section 1.2. For a general connected network, a tree physical graph can also be regarded as a subgraph of the original physical network topology, which would arise from restrictions imposed by spanning-tree based routing mechanisms [33] etc. Furthermore, different services could be assigned on different tree subgraphs of the physical network, which makes this assumption less restrictive.

In the tree application-graph topology, if we consider any path from the root to a leaf, we only allow those assignments where the application nodes along this path are assigned in their respective order on a sub-path of the physical topology (multiple application nodes may still be placed onto one physical node), thus, creating a "cycle-free" placement. Figure 3.1 illustrates this placement. Let nodes 1 to 5 denote the application nodes along a path in the application-graph topology. The cycle-free placement of this path onto a sub-path of the physical network ensures the order is preserved (as shown in Fig. 3.1(b)), whereas the order is not preserved in Fig. 3.1(c). A cycle-free placement has a clear motivation of avoiding cyclic commu-

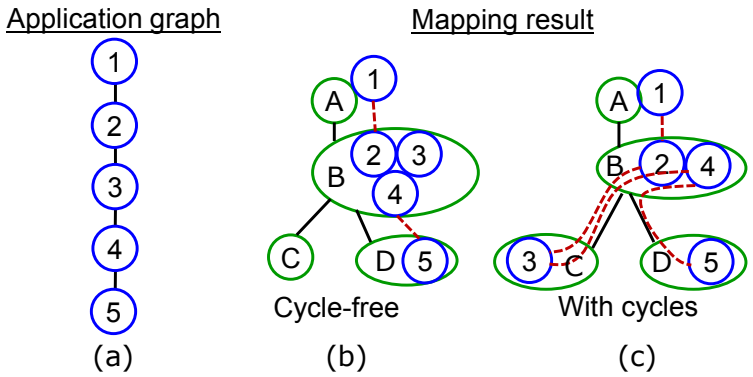


Figure 3.1: Mapping with and without cycles. In this example, the path in the application graph is between application node 1 and application node 5.

nication among the application nodes. For example, for the placement in Fig. 3.1(c), application nodes 2 and 4 are placed on physical node B, while application node 3 is placed on physical node C. In this case, the physical link B–C carries the data demand of application links 2–3 and 3–4 in a circular fashion. Such traffic can be naturally avoided with a cycle-free mapping (Fig. 3.1(b)), thus relieving congestion on the communication links. As we will see in the simulations in Section 3.5, the cycle-free constraint still allows the proposed scheme to outperform some other comparable schemes that allow cycles. Further discussion on the approximation ratio associated with the cycle-free restriction is given in Appendix A.

In this chapter, for the purpose of describing the algorithms, we classify an application node as a *junction node* in the application graph when it is a root node and is connected to two or more edges, or when it is *not* a root node and is connected to three or more edges (among which one edge connects to its parent node). These junction nodes can be more significant than other nodes, because they represent data splitting or joining processes for multiple data streams. In some cases, the junction nodes may have pre-specified placement, because they serve multiple data streams that may be associated with different end-users, and individual data streams may arrive dynamically in an online fashion. Our work first considers cases where the placements of these junction nodes are pre-specified, and then extends the results to

the general case where some junction nodes are not placed beforehand.

For the aforementioned scenarios, we obtain the following main results for the problem of service placement with the goal of load balancing among the physical nodes and edges:

1. An optimal offline algorithm for placing a single application graph which is a linear graph, with $O(V^3N^2)$ time-complexity and $O(VN(V+N))$ space-complexity, where the application graph has V nodes and the physical graph has N nodes.

2. An online approximation algorithm for placing single or multiple tree application graphs, in which the placements of all junction nodes are pre-specified, i.e., their placements are given. This algorithm has a time-complexity of $O(V^3N^2)$ and a space-complexity of $O(VN(V+N))$ for each application graph placement, and its competitive ratio is $O(\log N)$.

3. An online approximation algorithm for placing single or multiple tree application graphs, in which the placements of some junction nodes are *not* pre-specified. This algorithm has a time-complexity of $O(V^3N^{2+H})$ and a space-complexity of $O(VN^{1+H}(V+N))$ for each application graph placement, and its competitive ratio is $O(\log^{1+H} N)$, where H is the maximum number of junction nodes without given placement on any single path from the root to a leaf in the application graph. Note that we always have $H \leq D$, where D is the depth of the tree application graph.

We consider undirected application and physical graphs in this chapter, which means that data can flow in any direction on an edge, but the proposed algorithms can be easily extended to some types of directed graphs. For example, when the tree application graph is directed and the tree physical graph is undirected, we can merge the two application edges that share the same end nodes in different directions into

one edge, and focus on the merged undirected application graph for the purpose of finding optimal placement. This does not affect the optimality because for any placement of application nodes, there is a unique path connecting two different application nodes due to the cycle-free constraint and the tree structure of physical graphs. Thus, application edges in both directions connecting the same pair of application nodes have to be placed along the same path on the physical graph.

Our work also considers multiple types of resources on each physical node, such as CPU, storage, and I/O resources. The proposed algorithms can also work with domain constraints which restrict the set of physical nodes that a particular application node can be assigned to. The exact algorithm for single line placement can also incorporate conflict constraints where some assignments are not allowed for a pair of adjacent application nodes that are connected by an application edge; such constraints naturally arise in practice due to security policies as discussed in Section 2.2.

3.2 Problem Formulation

3.2.1 Definitions

We consider the placement of application graphs onto a physical graph, where the application graphs represent services that may arrive in an online fashion. In this chapter, we reuse the definitions given in Section 2.1. In addition, we introduce some additional definitions as follows.

Graphs: Because we consider multiple application graphs in this chapter, we denote the tree application graph for the ith service arrival as $\mathcal{R}(i) = (\mathcal{V}(i), \mathcal{E}(i))$. Throughout this chapter, we define $V = |\mathcal{V}|$, $E = |\mathcal{E}|$, $N = |\mathcal{N}|$, and $L = |\mathcal{L}|$, where $|\cdot|$ denotes the number of elements in the corresponding set.

Costs: For the ith service, the weighted cost (where the weighting factor can serve as a normalization to the total resource capacity) for type $k \in \{1, 2, ..., K\}$ resource of placing v to n is denoted by $d_{v\to n,k}(i)$. Similarly, the weighted communication bandwidth cost of assigning e to l is denoted by $b_{e\to l}(i)$. This edge cost is also defined for a dummy link $l = (n, n)$, namely a non-existing link that connects the same node, to take into account the additional cost when placing two application nodes on one physical node. It is also worth noting that an application edge may be placed onto multiple physical links that form a path.

Remark: The cost of placing an application node (or edge) onto different physical nodes (or edges) can be different. This is partly because different physical nodes and edges may have different resource capacities, and therefore different weighting factors for cost computation. It can also be due to the domain and conflict constraints as mentioned earlier. If some mapping is not allowed, then we can set the corresponding mapping cost to infinity. Hence, our cost definitions allow us to model a wide range of access-control/security policies.

Mapping: A mapping is specified by $\pi : \mathcal{V} \to \mathcal{N}$. Because we consider tree physical graphs with the cycle-free restriction, there exists only one path between two nodes in the physical graph, and we use (n_1, n_2) to denote either the link or path between nodes n_1 and n_2. We use the notation $l \in (n_1, n_2)$ to denote that link l is included in path (n_1, n_2). The placement of nodes automatically determines the placement of edges.

In a successive placement of the 1st up to the ith service, each physical node $n \in \mathcal{N}$ has an aggregated weighted cost of

$$p_{n,k}(i) = \sum_{j=1}^{i} \sum_{v:\pi(v)=n} d_{v\to n,k}(j), \tag{3.1}$$

where the second sum is over all v that are mapped to n. Equation (3.1) gives the

total cost of type k resource requested by all application nodes that are placed on node n, upto the ith service. Similarly, each physical edge $l \in \mathcal{L}$ has an aggregated weighted cost of

$$q_l(i) = \sum_{j=1}^{i} \sum_{e=(v_1,v_2):(\pi(v_1),\pi(v_2))\ni l} b_{e\to l}(j), \tag{3.2}$$

where the second sum is over all application edges $e = (v_1, v_2)$ for which the path between the physical nodes $\pi(v_1)$ and $\pi(v_2)$ (which v_1 and v_2 are respectively mapped to) includes the link l.

3.2.2 Objective Function

The optimization objective in this chapter is load balancing for which the objective function is defined as

$$\min_{\pi} \max \left\{ \max_{k,n} p_{n,k}(M); \max_{l} q_l(M) \right\}, \tag{3.3}$$

where M is the total number of services (application graphs). The function in (3.3) aims to minimize the maximum weighted cost on each physical node and link, ensuring that no single element gets overloaded and becomes a point of failure, which is important especially in the presence of bursty traffic. Such an objective has been widely used in the literature [34, 35].

While we choose the objective function (3.3) in this chapter, we do realize that there can be other objectives as well, such as minimizing the total resource consumption. We note that the exact algorithm for the placement of a single linear application graph can be generalized to a wide class of other objective functions as will be discussed in Section 3.3.5, but for simplicity, we restrict our attention to the objective function in (3.3) in the following discussion.

A Note on Capacity Limit: For simplicity, we do not impose capacity constraints on physical nodes and links in the optimization problem defined in (3.3), because even without the capacity constraint, the problem is very hard as we will see later in this chapter. However, because the resource demand of each application node and link is specified in every application graph, the total resource consumption at a particular physical node/link can be calculated by summing up the resource demands of application nodes/links that are placed on it. Therefore, an algorithm can easily check within polynomial time whether the current placement violates the capacity limits. If such a violation occurs, it can simply reject the newly arrived application graph.

In most practical cases, the costs of node and link placements should be defined as proportional to the resource occupation when performing such placement, with weights inversely proportional to the capacity of the particular type of resource, such as the cost definitions in Section 2.2. With such a definition, the objective function (3.3) essentially tries to place as many application graphs as possible without increasing the maximum resource occupation (normalized by the resource capacity) among all physical nodes and links. Thus, the placement result should utilize the available resource reasonably well. A more rigorous analysis on the impact of capacity limit is left as future work.

3.3 Basic Assignment Unit: Single Linear Application Graph Placement

We first consider the assignment of a single linear application graph (i.e., the application nodes are connected in a line), where the goal is to find the best placement of application nodes onto a path in the tree physical graph under the cycle-free constraint (as shown in Fig. 3.1). The solution to this problem forms the building block

of other more sophisticated algorithms presented later. As discussed next, we develop an algorithm that can find the optimal solution to this problem. We omit the service index i in this section because we focus on a single service, i.e. $M = 1$, here.

3.3.1 Problem Formulation

Without loss of generality, we assume that $\mathcal{V}$ and $\mathcal{N}$ are indexed sets, and we use v to exchangeably denote elements and indices of application nodes in $\mathcal{V}$, and use n to exchangeably denote elements and indices of physical nodes in $\mathcal{N}$. This index (starting from 1 for the root node) is determined by the topology of the graph. In particular, it can be determined via a breadth-first or depth-first indexing on the tree graph (note that linear graphs are a special type of tree graphs). From this it follows that, if n_1 is a parent of n_2, then we must have $n_1 < n_2$. The same holds for the application graph with nodes $\mathcal{V}$.

With this setting, the edge cost can be combined together with the cycle-free constraint into a single definition of pairwise costs. The weighted pairwise cost of placing $v-1$ to n_1 and v to n_2 is denoted by $c_{(v-1,v)\to(n_1,n_2)}$, and it takes the following values with $v \geq 2$:

1. If the path from n_1 to n_2 traverses some $n < n_1$, in which case the cycle-free assumption is violated, then $c_{(v-1,v)\to(n_1,n_2)} = \infty$.

2. Otherwise,

$$c_{(v-1,v)\to(n_1,n_2)} = \max_{l\in(n_1,n_2)} b_{(v-1,v)\to l}\Big|_{(\pi(v-1),\pi(v))\ni l}. \tag{3.4}$$

The maximum operator in (3.4) follows from the fact that, in the single line placement, at most one application edge can be placed onto a physical link. Also recall that the edge cost definition incorporates dummy links such as $l = (n, n)$, thus there always exists $l \in (n_1, n_2)$ even if $n_1 = n_2$.

Then, the optimization problem (3.3) with $M = 1$ becomes

$$\min_{\pi} \max \left\{ \max_{k,n} \sum_{v:\pi(v)=n} d_{v\to n,k}; \max_{(v-1,v)\in\mathcal{E}} c_{(v-1,v)\to(\pi(v-1),\pi(v))} \right\}. \tag{3.5}$$

The last maximum operator in (3.5) takes the maximum among all application edges (rather than physical links), because when combined with the maximum in (3.4), it essentially computes the maximum among all physical links that are used for data transmission under the mapping π.

3.3.2 Decomposing the Objective Function

In this subsection, we decompose the objective function in (3.5) to obtain an iterative solution. Note that the objective function (3.5) already incorporates all the constraints as discussed earlier. Hence, we only need to focus on the objective function itself.

When only considering a subset of application nodes $1, 2, ..., v_1 \leq V$, for a given mapping π, the value of the objective function for this subset of application nodes is

$$J_\pi(v_1) \;=\; \max \left\{ \max_{k,n} \sum_{v\leq v_1:\pi(v)=n} d_{v\to n,k}; \max_{(v-1,v)\in\mathcal{E}, v\leq v_1} c_{(v-1,v)\to(\pi(v-1),\pi(v))} \right\}. \tag{3.6}$$

Compared with (3.5), the only difference in (3.6) is that we consider the first v_1 application nodes and the mapping π is assumed to be given. The optimal cost for application nodes $1, 2, ..., v_1 \leq V$ is then

$$J_{\pi^*}(v_1) = \min_{\pi} J_\pi(v_1), \tag{3.7}$$

where π^* denotes the optimal mapping.

Proposition 3.1. ***(Decomposition of the Objective Function)****: Let* $J_{\pi^*|\pi(v_1)}(v_1)$ *denote the optimal cost under the condition that* $\pi(v_1)$ *is given, i.e.* $J_{\pi^*|\pi(v_1)}(v_1) = \min_{\pi(1),...,\pi(v_1-1)} J_\pi(v_1)$ *with given* $\pi(v_1)$*. When* $\pi(v_1) = \pi(v_1 - 1) = ... = \pi(v_s) > \pi(v_s - 1) \geq \pi(v_s - 2) \geq ... \geq \pi(1)$*, where* $1 \leq v_s \leq v_1$*, which means that* v_s *is mapped to a different physical node from* $v_s - 1$ *and nodes* $v_s, ..., v_1$ *are mapped onto the same physical node*[3]*, then we have*

$$J_{\pi^*|\pi(v_1)}(v_1) = \min_{v_s=1,...,v_1} \min_{\pi(v_s-1)} \max \Bigg\{ J_{\pi^*|\pi(v_s-1)}(v_s - 1); \max_{k=1,...,K} \sum_{v=v_s...v_1} d_{v\to\pi(v_1),k}; \max_{(v-1,v)\in\mathcal{E}, v_s\leq v\leq v_1} c_{(v-1,v)\to(\pi(v-1),\pi(v))} \Bigg\}. \tag{3.8}$$

The optimal mapping for v_1 *can be found by*

$$J_{\pi^*}(v_1) = \min_{\pi(v_1)} J_{\pi^*|\pi(v_1)}(v_1). \tag{3.9}$$

Proof. Because $\pi(v_s) = \pi(v_s + 1) = ... = \pi(v_1)$, we have

$$J_\pi(v_1) = \max \Bigg\{ J_\pi(v_s - 1); \max_{k=1,...,K} \sum_{v=v_s...v_1} d_{v\to\pi(v_1),k}; \max_{(v-1,v)\in\mathcal{E}, v_s\leq v\leq v_1} c_{(v-1,v)\to(\pi(v-1),\pi(v))} \Bigg\}. \tag{3.10}$$

The three terms in the maximum operation in (3.10) respectively correspond to: 1) the cost at physical nodes and edges that the application nodes $1, ..., v_s - 1$ (and their connecting edges) are mapped to, 2) the costs at the physical node that $v_s, ..., v_1$ are mapped to, and 3) the pairwise costs for connecting $v_s - 1$ and v_s as well as

[3]Note that when $v_s = 1$, then $v_s - 1$ does not exist, which means that all nodes $1, ..., v_1$ are placed onto the same physical node. For convenience, we define $J_\pi(0) = 0$.

interconnections[4] of nodes in $v_s, ..., v_1$. Taking the maximum of these three terms, we obtain the cost function in (3.6).

In the following, we focus on finding the optimal mapping based on the cost decomposition in (3.10). We note that the pairwise cost between $v_s - 1$ and v_s depends on the placements of both $v_s - 1$ and v_s. Therefore, in order to find the optimal $J_\pi(v_1)$ from $J_\pi(v_s-1)$, we need to find the minimum cost among all possible placements of $v_s - 1$ and v_s, provided that nodes $v_s, ..., v_1$ are mapped onto the same physical node and v_s and $v_s - 1$ are mapped onto different physical nodes. For a given v_1, node v_s may be any node that satisfies $1 \leq v_s \leq v_1$. Therefore, we also need to search through all possible values of v_s. This can then be expressed as the proposition, where we first find $J_{\pi^*|\pi(v_1)}(v_1)$ as an intermediate step. □

Equation (3.8) is the Bellman's equation [36] for problem (3.5). Using dynamic programming [36], we can solve (3.5) by iteratively solving (3.8). In each iteration, the algorithm computes new costs $J_{\pi^*|\pi(v_1)}(v_1)$ for all possible mappings $\pi(v_1)$, based on the previously computed costs $J_{\pi^*|\pi(v)}(v)$ where $v < v_1$. For the final application node $v_1 = V$, we use (3.9) to compute the final optimal cost $J_{\pi^*}(V)$ and its corresponding mapping π^*.

3.3.3 Optimal Algorithm

The pseudocode of the exact optimal algorithm is shown in Algorithm 3.1. It computes $V \cdot N$ number of $J_{\pi^*|\pi(v)=n}(v)$ values, and we take the minimum among no more than $V \cdot N$ values in (3.8). The terms in (3.8) include the sum or maximum of no more than V values and the maximum of K values. Because K is a con-

[4]Note that, although $v_s, ..., v_1$ are mapped onto the same physical node, their pairwise costs may be non-zero if there exists additional cost when placing different application nodes onto the same physical node. In the extreme case where adjacent application nodes are not allowed to be placed onto the same physical node (i.e. conflict constraint), their pairwise cost when placing them on the same physical node becomes infinity.

Algorithm 3.1 Placement of a linear application graph onto a tree physical graph

1: Given linear application graph $\mathcal{R}$, tree physical graph $\mathcal{Y}$
2: Given $V \times N \times K$ matrix $\mathbf{D}$, its entries represent the weighted type k node cost $d_{v \to n,k}$
3: Given $(V-1) \times N \times N$ matrix $\mathbf{C}$, its entries represent the weighted pairwise cost $c_{(v-1,v)\to(n_1,n_2)}$
4: Define $V \times N$ matrix $\mathbf{J}$ to keep the costs $J_{\pi^*|\pi(v)=n}(v)$ for each node (v,n) in the auxiliary graph
5: Define $V \times N \times V$ matrix $\mathbf{\Pi}$ to keep the mapping corresponding to its cost $J_{\pi^*|\pi(v)=n}(v)$ for each node (v,n) in the auxiliary graph
6: **for** $v = 1...V$ **do**
7: **for** $n = 1...N$ **do**
8: Compute $J_{\pi^*|\pi(v)=n}(v)$ from (3.8), put the result into $\mathbf{J}$ and the corresponding mapping into $\mathbf{\Pi}$
9: **end for**
10: **end for**
11: Compute $J_{\pi^*}(V) \leftarrow \min_n J_{\pi^*|\pi(V)=n}(V)$
12: **return** the final mapping result π^* and final optimal cost $J_{\pi^*}(V)$

stant in practical systems, we conclude that the *time-complexity of this algorithm is* $O(V^3N^2)$.

The *space-complexity of Algorithm 3.3 is* $O(VN(V+N))$, which is related to the memory required for storing matrices $\mathbf{D}$, $\mathbf{C}$, $\mathbf{J}$, and $\mathbf{\Pi}$ in the algorithm, where K is also regarded as a constant here.

Also note that the optimality of the result from Algorithm 3.1 is subject to the cycle-free constraint, and the sequence of nodes is always preserved in each iteration.

3.3.4 Example

To illustrate the procedure of the algorithm, we construct an auxiliary graph from the given application and physical graphs, as shown in Fig. 3.2. Each node (v_1, n_1) in the auxiliary graph represents a possible placement of a particular application node, and is associated with the cost value $J_{\pi^*|\pi(v_1)=n_1}(v_1)$, where v_1 is the application node index and n_1 is the physical node index in the auxiliary graph. When computing the cost at a particular node, e.g. the cost $J_{\pi^*|\pi(4)=\mathrm{C}}(4)$ at node (4,C) in Fig. 3.2, the

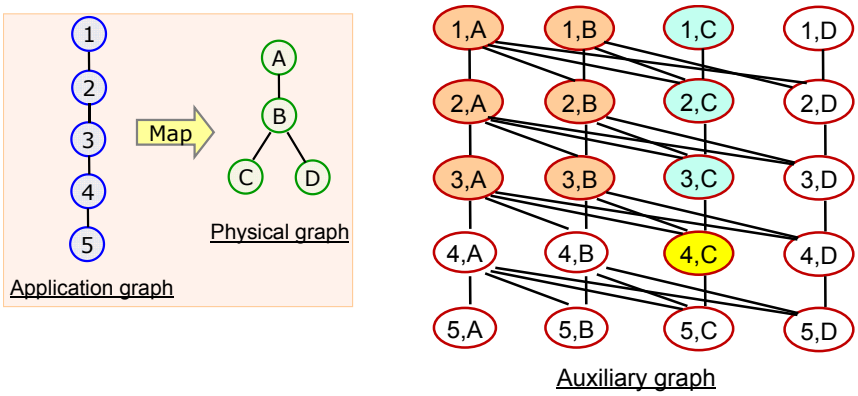


Figure 3.2: Auxiliary graph and algorithm procedure for the placement of a linear application graph onto a tree physical graph.

algorithm starts from the "earlier" costs $J_{\pi^*|\pi(v_s-1)}(v_s-1)$ where the tuple $(v_s-1, \pi(v_s-1))$ is either (1,A), (1,B), (2,A), (2,B), (3,A), or (3,B). From each of these nodes, the subsequent application nodes (i.e. from v_s to node 4) are all mapped onto physical node C, and we compute the cost for each such "path" with the maximum operations in (3.8), by assuming the values of v_s-1 and $\pi(v_s-1)$ are given by its originating node in the auxiliary graph. For example, one path can be (2,B) – (3,C) – (4,C) where $v_s-1=2$ and $\pi(v_s-1)$ = B, another path can be (1,A) – (2,C) – (3,C) – (4,C) where $v_s-1=1$ and $\pi(v_s-1)$ = A. Then, the algorithm takes the minimum of the costs for all paths, which corresponds to the minimum operations in (3.8) and gives $J_{\pi^*|\pi(4)=\mathrm{C}}(4)$. In the end, the algorithm searches through all the possible mappings for the final application node (node 5 in Fig. 3.2) and chooses the mapping that results in the minimum cost, which corresponds to the procedure in (3.9).

3.3.5 Extensions

The placement algorithm for single linear application graph can also be used when the objective function (in the form of (3.3) with $M=1$) is modified to one of the

following:

$$\min_{\pi} \max \left\{ \max_{k,n} f_{n,k} \left(\sum_{v:\pi(v)=n} d_{v\to n,k} \right) ; \max_{l} g_l \left(\sum_{e=(v_1,v_2):(\pi(v_1),\pi(v_2))\ni l} b_{e\to l} \right) \right\} \tag{3.11}$$

or

$$\min_{\pi} \sum_{k,n} f_{n,k} \left(\sum_{v:\pi(v)=n} d_{v\to n,k} \right) + \sum_{l} g_l \left(\sum_{e=(v_1,v_2):(\pi(v_1),\pi(v_2))\ni l} b_{e\to l} \right), \tag{3.12}$$

where $f_{n,k}(\cdot)$ and $g_l(\cdot)$ are increasing functions with $f_{n,k}(0) = 0$, $g_l(0) = 0$, $f_{n,k}(\infty) = \infty$, and $g_l(\infty) = \infty$. The algorithm and its derivation follow the same procedure as discussed above. These alternative objective functions can be useful for scenarios where the goal of optimization is other than min-max. The objective function in (3.12) will also be used later for solving the online placement problem.

3.4 Online Placement Algorithms for Tree Application Graphs

Using the optimal algorithm for the single linear application graph placement as a sub-routine, we now present algorithms for the generalized case; namely, placement of an arriving stream of application graphs with tree topology. We first show that even the offline placement of a single tree is NP-hard. Then, we propose online algorithms to approximate the optimal placement with provable competitive ratio, by first considering the case where junction nodes in the application graph have pre-specified placements that are given beforehand, and later relax this assumption.

3.4.1 Hardness Result

Proposition 3.2. *(**NP-hardness**) Placement of a tree application graph onto a tree physical graph for the objective function defined in (3.3), with or without pre-specified junction node placement, is NP-hard.*

Proof. To show that the given problem is NP-hard, we show that the problem can be reduced from the NP-hard problem of minimum makespan scheduling on unrelated parallel machines (MMSUPM) [23], which minimizes the maximum load (or job processing time) on each machine.

Consider a special case in our problem where the application graph has a star topology with two levels (one root and multiple leaf nodes), and the physical graph is a line with multiple nodes. Assume that the number of resource types in the nodes is $K = 1$, the application edge resource demand is zero, and the application node resource demand is non-zero. Then, the problem is essentially the MMSUPM problem. It follows that the MMSUPM problem reduces to our problem. In other words, if we can solve our problem in polynomial time, then we can also solve the MMSUPM problem in polynomial time. Because MMSUPM is NP-hard, our problem is also NP-hard. The above result holds no matter whether the root node (junction node) of the application graph has pre-specified placement or not. □

Note that although the objective functions (3.3) and (2.3) are slightly different, the above proof essentially shows that the problem in (2.3) is also NP-hard. This is because when setting application edge resource demands to zero as done in the proof, the problems (3.3) and (2.3) become the same.

3.4.2 When All Junction Node Placements Are Given

We consider tree application graphs for which the placements of junction nodes are given, and focus on placing the remaining non-junction nodes which are connected

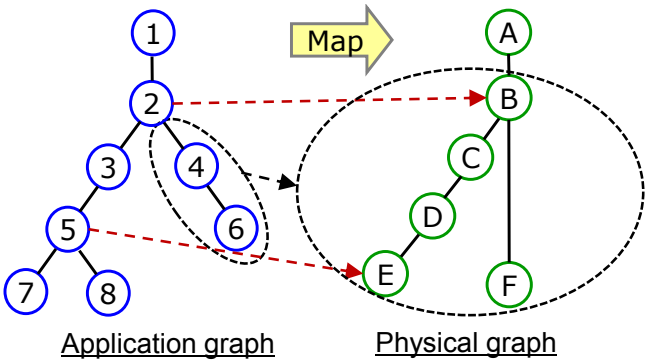


Figure 3.3: Example of application graph with given placement of junction nodes. Junction node 2 is placed on physical node B and junction node 5 is placed on physical node E. The algorithm needs to decide the placement of the remaining nodes, subject to the cycle-free constraint.

to at most two edges. An example is shown in Fig. 3.3. Given the placed junction nodes, we name the set of application edges and nodes that form a chain between the placed nodes (excluding each placed node itself, but including each edge that is connected to a placed node) as a *simple branch*, where the notion "simple" is opposed to the general branch which will be defined in Section 3.4.3. A simple branch can also be a chain starting from an edge that connects a placed node and ending at a leaf node, such as the nodes and edges within the dashed boundary in the application graph in Fig. 3.3. Each node in a simple branch is connected to at most two edges.

3.4.2.1 Algorithm Design

In the following, we propose an online placement algorithm, where we make use of some ideas from [37]. Different from [37] which focused on routing and job scheduling problems, our work considers more general graph mapping.

When a service (represented by a tree application graph) arrives, we *split* the whole application graph into simple branches, and regard each simple branch as an independent application graph. All the nodes with given placement can also be regarded as an application/service that is placed before placing the individual simple

branches. After placing those nodes, each individual simple branch is placed using the online algorithm that we describe below. In the remaining of this section, by service we refer to the *service after splitting*, i.e. each service either consists of a simple branch or a set of nodes with given placement.

How to Place Each Simple Branch: While our ultimate goal is to optimize (3.3), we use an *alternative objective function* to determine the placement of each newly arrived service i (after splitting). Such an indirect approach provides performance guarantee with respect to (3.3) in the long run. We will first introduce the new objective function and then discuss its relationship with the original objective function (3.3).

We first define a variable $\hat{J}$ as a reference cost. The reference cost may be an estimate of the true optimal cost (defined as in (3.3)) from the optimal offline placement, and the method to determine this value will be discussed later. Then, for placing the ith service, we use an objective function which has the same form as (3.12), with $f_{n,k}(\cdot)$ and $g_l(\cdot)$ defined as

$$f_{n,k}(x) \triangleq \exp_\alpha\left(\frac{p_{n,k}(i-1)+x}{\hat{J}}\right) - \exp_\alpha\left(\frac{p_{n,k}(i-1)}{\hat{J}}\right), \quad (3.13a)$$

$$g_l(x) \triangleq \exp_\alpha\left(\frac{p_l(i-1)+x}{\hat{J}}\right) - \exp_\alpha\left(\frac{p_l(i-1)}{\hat{J}}\right), \quad (3.13b)$$

subject to the cycle-free placement constraint, where we define $\exp_\alpha(y) \triangleq \alpha^y$ and $\alpha \triangleq 1 + 1/\gamma$, in which $\gamma > 1$ is a design parameter.

Why We Use an Alternative Objective Function: The objective function (3.12) with (3.13a) and (3.13b) is the increment of the sum exponential values of the original costs, given all the previous placements. With this objective function, the performance bound of the algorithm can be shown analytically (see Proposition 3.3 below). Intuitively, the new objective function (3.12) serves the following purposes:

1. "Guide" the system into a state such that the maximum cost among physical links and nodes is not too high, thus approximating the original objective function (3.3). This is because when the existing cost at a physical link or node (for a particular resource type k) is high, the incremental cost (following (3.12)) of placing the new service i on this link or node (for the same resource type k) is also high, due to the fact that $\exp_\alpha(y)$ is convex increasing and the cost definitions in (3.13a) and (3.13b).

2. While (3.3) only considers the maximum cost, (3.12) is also related to the sum cost, because we sum up all the exponential cost values at different physical nodes and links together. This "encourages" a low resource consuming placement of the new service i (which is reflected by low sum cost), thus leaving more available resources for future services. In contrast, if we use (3.3) directly for each newly arrived service, the placement may greedily take up too much resource, so that future services can no longer be placed with a low cost.

In practice, we envision that objective functions with a shape similar to (3.12) can also serve our purpose.

How to Solve It: Because each service either obeys a pre-specified placement or consists of a simple branch, we can use Algorithm 3.1 with appropriately modified cost functions to find the optimal solution to (3.12) with (3.13a) and (3.13b). For the case of a simple branch having an open edge, such as edge $(2, 4)$ in Fig. 3.3, we connect an application node that has zero resource demand to extend the simple branch to a graph, so that Algorithm 3.1 is applicable.

Algorithm 3.2 summarizes the above argument as a formal algorithm for each service placement, where π_i denotes the mapping for the ith service. Define a parameter, $\beta = \log_\alpha \left(\frac{\gamma(NK+L)}{\gamma-1}\right)$, then Algorithm 3.2 performs the placement as long as the cost on each node and link is not bigger than $\beta\hat{J}$, otherwise it returns FAIL. The significance of the parameter β is in calculating the competitive ratio, i.e. the

Algorithm 3.2 Online placement of a service that is either a simple branch or a set of nodes with given placement

Given the ith service that is either a set of nodes with given placement or a simple branch
Given tree physical graph $\mathcal{Y}$
Given $p_{n,k}(i-1)$, $q_l(i-1)$, and placement costs
Given $\hat{J}$ and β
if service is a set of nodes with given placement **then**
    Obtain π_i based on given placement
else if service is a simple branch **then**
    Extend simple branch to linear graph $\mathcal{R}(i)$, by connecting zero-resource-demand application nodes to open edges, and the placements of these zero-resource-demand application nodes are given
    Run Algorithm 3.1 with objective function (3.12) with (3.13a) and (3.13b), for $\mathcal{R}(i)$, to obtain π_i
end if
if $\exists n,k \ : \ p_{n,k}(i-1) + \sum_{v:\pi_i(v)=n} d_{v\to n,k}(i) > \beta\hat{J}$ or $\exists l \ : \ q_l(i-1) + \sum_{e=(v_1,v_2):(\pi_i(v_1),\pi_i(v_2))\ni l} b_{e\to l}(i) > \beta\hat{J}$ **then**
    return FAIL
else
    return π_i
end if

maximum ratio of the cost resulting from Algorithm 3.2 to the optimal cost from an equivalent offline placement, as shown in Proposition 3.3.

Why We Need the Reference Cost $\hat{J}$: The reference cost $\hat{J}$ is an input parameter of the objective function (3.12) and Algorithm 3.2, which enables us to show a performance bound for Algorithm 3.2, as shown in Proposition 3.3.

Proposition 3.3. *If there exists an offline mapping π^o that considers all M application graphs and brings cost J_{π^o}, such that $J_{\pi^o} \leq \hat{J}$, then Algorithm 3.2 never fails, i.e., $p_{n,k}(M)$ and $q_l(M)$ from Algorithm 3.2 never exceeds $\beta\hat{J}$. The cost J_{π^o} is defined in (3.3).*

Proof. See Appendix C.1. □

Proposition 3.3 guarantees a bound for the cost resulting from Algorithm 3.2. We note that the optimal offline mapping π^{o*} produces cost $J_{\pi^{o*}}$, which is smaller

than or equal to the cost of an arbitrary offline mapping. It follows that for any π^o, we have $J_{\pi^{o*}} \leq J_{\pi^o}$. This means that if there exists π^o such that $J_{\pi^o} \leq \hat{J}$, then we must have $J_{\pi^{o*}} \leq \hat{J}$. If we can set $\hat{J} = J_{\pi^{o*}}$, then from Proposition 3.3 we have $\max\{\max_{k,n} p_{n,k}(M); \max_l q_l(M)\} \leq \beta J_{\pi^{o*}}$, which means that the competitive ratio is β.

How to Determine the Reference Cost $\hat{J}$: Because the value of $J_{\pi^{o*}}$ is unknown, we cannot always set $\hat{J}$ exactly to $J_{\pi^{o*}}$. Instead, we need to set $\hat{J}$ to an estimated value that is not too far from $J_{\pi^{o*}}$. We achieve this by using the doubling technique, which is widely used in online approximation algorithms. The idea is to double the value of $\hat{J}$ every time Algorithm 3.2 fails. After each doubling, we ignore all the previous placements when calculating the objective function (3.12) with (3.13a) and (3.13b), i.e. we assume that there is no existing service, and we place the subsequent services (including the one that has failed with previous value of $\hat{J}$) with the new value of $\hat{J}$. At initialization, the value of $\hat{J}$ is set to a reasonably small number $\hat{J}_0$.

In Algorithm 3.3, we summarize the high-level procedure that includes the splitting of the application graph, the calling of Algorithm 3.2, and the doubling process, with multiple application graphs that arrive over time.

3.4.2.2 Complexity and Competitive Ratio

In the following, we discuss the complexity and competitive ratio of Algorithm 3.3.

Because the value of $J_{\pi^{o*}}$ is finite[5], the doubling procedure in Algorithm 3.3 only contains finite steps. The remaining part of the algorithm mainly consists of calling Algorithm 3.2 which then calls Algorithm 3.1 for each simple branch. Because nodes

[5]The value of $J_{\pi^{o*}}$ is finite unless the placement cost specification does not allow any placement with finite cost. We do not consider this case here because it means that the placement is not realizable under the said constraints. In practice, the algorithm can simply reject such application graphs when the mapping cost resulting from Algorithm 3.2 is infinity, regardless of what value of $\hat{J}$ has been chosen.

Algorithm 3.3 High-level procedure for multiple arriving tree application graphs

Initialize $\hat{J} \leftarrow \hat{J}_0$
Define index i as the service index, which automatically increases by 1 for each new service (after splitting)
Initialize $i \leftarrow 1$
Initialize $i_0 \leftarrow 1$
loop
  if new application graph has arrived **then**
    Split the application graph into simple branches and a set of nodes with given placement, assume that each of them constitute a service
    for all service i **do**
      repeat
        Call Algorithm 3.2 for service i with
        $p_{n,k}(i-1) = \max\{0, p_{n,k}(i-1) - p_{n,k}(i_0-1)\}$ and
        $q_l(i-1) = \max\{0, q_l(i-1) - q_l(i_0-1)\}$
        if Algorithm 3.2 returns FAIL **then**
          Set $\hat{J} \leftarrow 2\hat{J}$
          Set $i_0 \leftarrow i$
        end if
      until Algorithm 3.2 does not return FAIL
      Map service i according to π_i resulting from Algorithm 3.2
    end for
  end if
end loop

and links in each simple branch together with the set of nodes with given placement add up to the whole application graph, similar to Algorithm 3.1, the *time-complexity of Algorithm 3.3 is* $O(V^3N^2)$ for each application graph arrival.

Similarly, when combining the procedures in Algorithms 3.1–3.3, we can see that the *space-complexity of Algorithm 3.3 is* $O(VN(V+N))$ for each application graph arrival, which is in the same order as Algorithm 3.1.

For the competitive ratio, we have the following result.

Proposition 3.4. ***(Competitive Ratio)****: Algorithm 3.3 is* $4\beta = 4\log_\alpha\left(\frac{\gamma(NK+L)}{\gamma-1}\right)$*-competitive.*

Proof. If Algorithm 3.2 fails, then we know that $J_{\pi^{o*}} > \hat{J}$ according to Proposition 3.3. Hence, by doubling the value of $\hat{J}$ each time Algorithm 3.2 fails, we have $\hat{J}_f < 2J_{\pi^{o*}}$, where $\hat{J}_f$ is the final value of $\hat{J}$ after placing all M services. Because we ignore all previous placements and only consider the services $i_0, ..., i$ for a particular value of $\hat{J}$, it follows that

$$\max\left\{\max_{k,n}\{p_{n,k}(i) - p_{n,k}(i_0-1)\}; \max_{l}\{q_l(i) - q_l(i_0-1)\}\right\} \leq \beta\hat{J} \tag{3.14}$$

for the particular value of $\hat{J}$.

When we consider all the placements of M services, by summing up (3.14) for all values of $\hat{J}$, we have

$$\begin{aligned}
&\max\left\{\max_{k,n} p_{n,k}(M); \max_{l} q_l(M)\right\} \\
\leq\ & \left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots\right)\beta\hat{J}_f \\
<\ & 2\left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots\right)\beta J_{\pi^{o*}} \\
=\ & 4\beta J_{\pi^{o*}}.
\end{aligned}$$

Hence, the proposition follows. □

The variables α, γ and K are constants, and $L = N - 1$ because the physical graph is a tree. Hence, the competitive ratio of Algorithm 3.3 can also be written as $O(\log N)$.

It is also worth noting that, for each application graph, we may have different tree physical graphs that are extracted from a general physical graph, and the above conclusions still hold.

3.4.3 When at Least One Junction Node Placement Is Not Given

In this subsection, we focus on cases where the placements of some or all junction nodes are not given. For such scenarios, we first extend our concept of branches to incorporate some unplaced junction nodes. The basic idea is that each *general branch* is the largest subset of nodes and edges that are interconnected with each other not including any of the nodes with pre-specified placement, but (as with our previous definition of simple branches) the subset includes the edges connected to placed nodes. A simple branch (see definition in Section 3.4.2) is always a general branch, but a general branch may or may not be a simple branch. Examples of general branches are shown in Fig. 3.4.

3.4.3.1 Algorithm Design

The main idea behind the algorithm is to combine Algorithm 3.2 with the enumeration of possible placements of unplaced junction nodes. When there is only a constant number of such nodes on any path from the root to a leaf, the algorithm remains polynomial in time-complexity while guaranteeing a poly-log competitive ratio.

To illustrate the intuition, consider the example application graph shown in Fig. 3.4(a), where nodes 2 and 5 are both initially unplaced. We follow a hierarchical

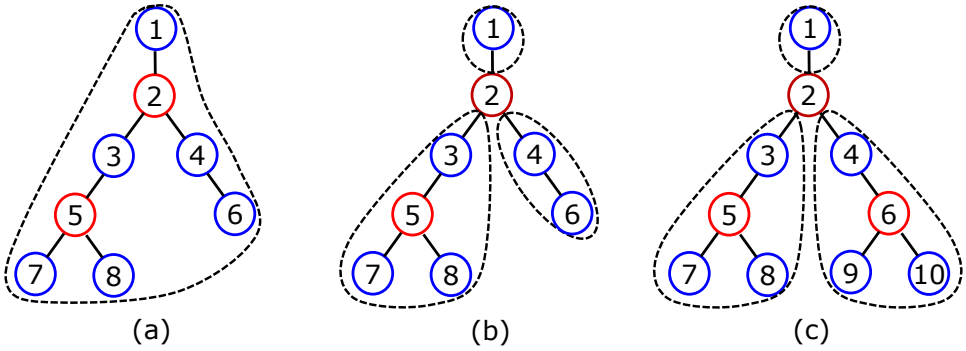


Figure 3.4: Example of application graphs with some unplaced junction nodes, the nodes and edges within each dashed boundary form a general branch: (a) nodes 2 and 5 are both unplaced, (b) node 2 is placed, node 5 is unplaced, (c) node 2 is placed, nodes 5 and 6 are unplaced.

determination of the placement of unplaced nodes starting with the nodes in the deepest level. For the example in Fig. 3.4(a), we first determine the placement of node 5, given each possible placement of node 2; and then determine the placement of node 2. Recall that we use the cost function in (3.12) with (3.13a) and (3.13b) to determine the placement of each simple branch when all the junction nodes are placed. We use the same cost function (with slightly modified parameters) for the placement of nodes 2 and 5. However, when determining the placement of node 5, we regard the general branch that includes node 5 (which contains nodes 3, 5, 7, and 8 and the corresponding edges as shown in Fig. 3.4(b)) as one single service, i.e. the values of $p_{n,k}(i-1)$ and $q_l(i-1)$ in (3.13a) and (3.13b) correspond to the resource utilization at nodes and links before placing this whole general branch, and the service i contains all the nodes and edges in this general branch. Similarly, when determining the placement of node 2, we consider the whole application graph as a single service.

It is worth noting that in many cases we may not need to enumerate all the possible combinations of the placement of unplaced junction nodes. For example, in Fig. 3.4(c), when the placement of node 2 is given, the placement of nodes 5 and 6 does not impose additional restrictions upon each other (i.e., the placement of node 5 does

not affect where node 6 can be placed, for instance). Hence, the general branches that respectively include node 5 and node 6 can be placed in a subsequent order using the online algorithm.

Based on the above examples, we summarize the procedure as Algorithm 3.4, where we solve the problem recursively and determine the placement of one junction node that has not been placed before in each instance of the function Unplaced(v, h). The parameter v is initially set to the top-most unplaced junction node (node 2 in Fig. 3.4(a)), and h is initially set to H (the maximum number of unplaced junction nodes on any path from the root to a leaf in the application graph).

Algorithm 3.4 can be embedded into Algorithm 3.3 to handle multiple arriving application graphs and unknown reference cost $\hat{J}$. The only part that needs to be modified in Algorithm 3.3 is that it now splits the whole application graph into general branches (rather than simple branches without unplaced junction nodes), and it either calls Algorithm 3.2 or Algorithm 3.4 depending on whether there are unplaced junction nodes in the corresponding general branch. When there are such nodes, it calls Unplaced(v, h) with the aforementioned initialization parameters.

3.4.3.2 Complexity and Competitive Ratio

The *time-complexity of Algorithm 3.4* together with its high-level procedure that is a modified version of Algorithm 3.3 is $O(V^3N^{2+H})$ for each application graph arrival, as explained below. Note that H is generally not the total number of unplaced nodes.

Obviously, when $H = 0$, the time-complexity is the same as the case where all junction nodes are placed beforehand. When there is only one unplaced junction node (in which case $H = 1$), Algorithm 3.4 considers all possible placements for this vertex, which has at most N choices. Hence, its time-complexity becomes N times the time-complexity with all placed junction nodes. When there are multiple unplaced junction nodes, we can see from Algorithm 3.4 that it only increases its re-

Algorithm 3.4 Tree-to-tree placement when some junction nodes are not placed

function Unplaced(v, h)
Given the ith service that is a general branch, tree physical graph $\mathcal{Y}$, $\hat{J}$, and β
Given $p_{n,k}(i-1)$ and $q_l(i-1)$ which is the current resource utilization on nodes and links
Define $\mathbf{\Pi}$ to keep the currently obtained mappings, its entry $\pi|_{\pi(v)=n_0}$ for all n_0 represents the mapping, given that v is mapped to n_0
Define $p_{n,k}(i)|_{\pi(v)=n_0}$ and $q_l(i)|_{\pi(v)=n_0}$ for all n_0 as the resource utilization after placing the ith service, given that v is mapped to n_0
Initialize $p_{n,k}(i)|_{\pi(v)=n_0} \leftarrow p_{n,k}(i-1)$ and $q_l(i)|_{\pi(v)=n_0} \leftarrow q_l(i-1)$ for all n_0
for all n_0 that v can be mapped to **do**
    Assume v is placed at n_0
    for all general branch that is connected with v **do**
        if the general branch contains unplaced junction nodes **then**
            Find the top-most unplaced vertex v' within this general branch
            Call Unplaced($v', h-1$) while assuming v is placed at n_0, and with $p_{n,k}(i-1) = p_{n,k}(i)|_{\pi(v)=n_0}$ and $q_l(i-1) = q_l(i)|_{\pi(v)=n_0}$
        else
            (in which case the general branch is a simple branch without unplaced junction nodes)
            Run Algorithm 3.2 for this branch
        end if
        Put mappings resulting from Unplaced($v', h-1$) or Algorithm 3.2 into $\pi|_{\pi(v)=n_0}$
        Update $p_{n,k}(i)|_{\pi(v)=n_0}$ and $q_l(i)|_{\pi(v)=n_0}$ to incorporate new mappings
    end for
end for
Find $\min_{n_0} \sum_{k,n} \left(\exp_\alpha \left(\frac{p_{n,k}(i)|_{\pi(v)=n_0}}{\beta^h \hat{J}} \right) - \exp_\alpha \left(\frac{p_{n,k}(i-1)}{\beta^h \hat{J}} \right) \right) + \sum_l \left(\exp_\alpha \left(\frac{q_l(i)|_{\pi(v)=n_0}}{\beta^h \hat{J}} \right) - \exp_\alpha \left(\frac{q_l(i-1)}{\beta^h \hat{J}} \right) \right)$, returning the optimal placement of v as n_0^*.
if $h = H$ and ($\exists n, k : p_{n,k}(i)|_{\pi(v)=n_0^*} > \beta^{1+H} \hat{J}$ or $\exists l : q_l(i)|_{\pi(v)=n_0^*} > \beta^{1+H} \hat{J}$) **then**
    return FAIL
else
    return $\pi|_{\pi(v)=n_0^*}$
end if

cursion depth when some lower level unplaced junction nodes exist. In other words, parallel general branches (such as the two general branches that respectively include node 5 and node 6 in Fig. 3.4(c)) do not increase the recursion depth, because the function Unplaced(v, h) for these general branches is called in a sequential order. Therefore, the time-complexity depends on the maximum recursion depth which is H; thus, the overall time-complexity is $O(V^3 N^{2+H})$.

The *space-complexity of Algorithm 3.4 is* $O(VN^{1+H}(V+N))$ for each application graph arrival, because in every recursion, the results for all possible placements of v are stored, and there are at most N such placements for each junction node.

Regarding the competitive ratio, similar to Proposition 3.3, we can obtain the following result.

Proposition 3.5. *If there exists an offline mapping* π^o *that considers all* M *application graphs and brings cost* J_{π^o}*, such that* $J_{\pi^o} \leq \hat{J}$*, then Algorithm 3.4 never fails, i.e.,* $p_{n,k}(M)$ *and* $q_l(M)$ *resulting from Algorithm 3.4 never exceeds* $\beta^{1+H}\hat{J}$*.*

Proof. When $H = 0$, the claim is the same as Proposition 3.3.

When $H = 1$, there is at most one unplaced junction node in each general branch. Because Algorithm 3.4 operates on each general branch, we can regard that we have only one unplaced junction node when running Algorithm 3.4. In this case, there is no recursive calling of Unplaced(v, h). Recall that v is the top-most unplaced junction node. The function Unplaced(v, h) first fixes the placement of this unplaced junction node v to a particular physical node n_0, and finds the placement of the remaining nodes excluding v. It then finds the placement of v.

From Proposition 3.3, we know that when we fix the placement of v, the cost resulting from the algorithm never exceeds $\beta\hat{J}$ if there exists a mapping $\pi^o|_{\pi(v)=n_0}$ (under the constraint that v is placed at n_0) that brings cost $J_{\pi^o|_{\pi(v)=n_0}}$ with $J_{\pi^o|_{\pi(v)=n_0}} \leq \hat{J}$.

To find the placement of v, Algorithm 3.4 finds the minimum cost placement

from the set of placements that have been obtained when the placement of v is given. Reapplying Proposition 3.3 for the placement of v, by substituting $\hat{J}$ with $\beta\hat{J}$, we know that the cost from the algorithm never exceeds $\beta^2\hat{J}$, provided that there exists a mapping, which is within the set of mappings produced by the algorithm with given v placements[6], that has a cost not exceeding $\beta\hat{J}$. Such a mapping exists and can be produced by the algorithm if there exists an offline mapping π^o (thus a mapping $\pi^o|_{\pi(v)=n_0}$ for a particular placement of v) that brings cost J_{π^o} with $J_{\pi^o} \leq \hat{J}$. Hence, the claim follows for $H = 1$.

When $H > 1$, because we decrease the value of h by one every time we recursively call Unplaced(v, h), the same propagation principle of the bound applies as for the case with $H = 1$. Hence, the claim follows. □

Using the same reasoning as for Proposition 3.4, it follows that *Algorithm 3.4 in combination with the extended version of Algorithm 3.3 is* $4\beta^{1+H} = 4\log_\alpha^{1+H}\left(\frac{\gamma(NK+L)}{\gamma-1}\right)$*-competitive, thus its competitive ratio is* $O(\log^{1+H} N)$.

3.5 Numerical Evaluation

In this section, we evaluate the performance of the proposed algorithm via simulation. For comparison, we consider online algorithms that utilize a mixed-integer linear program (MILP) solution. Specifically, a MILP optimization is solved for each new service arrival, and the service is placed according to the solution from the

[6]Note that, as shown in Line 20 of Algorithm 3.4, to determine the placement of v, we only take the minimum cost (expressed as the difference of exponential functions) with respect to those mappings that were obtained with given placement of v. It follows that the minimization is only taken among a subset of all the possible mappings. This restricts the reference mapping to be within the set of mappings that the minimization operator operates on. Because, only in this way, the inequality (C.5) in the proof of Proposition 3.3 can be satisfied. On the contrary, Algorithm 3.2 considers all possible mappings that a particular simple branch can be mapped to, by calling Algorithm 3.1 as its subroutine.

MILP. By service, we refer to non-split services, i.e. whole application graphs, in this section.

MILPs are generally *not* solvable in polynomial-time, but we use them for comparison to capture the *best possible result for a larger class of online algorithms* (including LP-relaxation or other heuristics) that place each service upon its arrival. Also note that these MILP results do *not* represent the optimal offline solution, because an optimal offline solution needs to consider all services at the same time, whereas the methods that we use for comparison only solve the MILP for each newly arrived service. Obtaining the optimal offline solution would require excessive computational time, hence we do not consider it here. We use the optimization toolbox CPLEX [20] to solve the MILPs in the simulation.

Recall that the goal of the proposed algorithm is to minimize the maximum resource utilization on nodes and links after hosting a number of services, as given in (3.3), and the (non-linear) exponential-difference cost function in (3.12) with (3.13a) and (3.13b) is used to determine the placement of each newly arrived service. In the methods for comparison, we also consider (3.3) as the "long-term" goal, but we consider two different MILP formulations with different objective functions for placing each newly arrived service. The first formulation greedily minimizes the maximum resource utilization for each service placement, with a min-max objective function. Such a formulation represents a general class of greedy mechanisms. The second formulation has a min-sum objective function as defined in [17]. The coefficients in the cost function are inversely proportional to the amount of available resource. Hence, this method also considers load balancing in its formulation. We do *not* impose the cycle-free constraint in the MILP formulations. When the placements of junction nodes are given, the children of this junction node can only be placed onto the physical node, on which the junction node has been placed, or onto the children of this physical node. This is considering that a placed junction node may be

responsible for nodes in a specific area.

We consider randomly generated tree application and physical graphs[7]. The number of application nodes for each service is randomly chosen from the interval $[3, 10]$, and the number of physical nodes ranges from 2 to 50. This setting is similar to that in related work such as [17]. We use a sequential approach to assign connections between nodes. Namely, we first label the nodes with indices. Then, we start from the lowest index, and connect each node m to those nodes that have indices $1, 2, ..., m-1$. Node m connects to node $m-1$ with probability 0.7, and connects to nodes $1, 2, ..., m-2$ each with probability $0.3/(m-2)$. We restrict the application root node to be placed onto the physical root node, which is due to the consideration that some portion of processing has to be performed on the core cloud in an MMC environment. We consider 100 service arrivals and simulate with 100 different random seeds to obtain the overall performance. The placement cost of a single node or link is uniformly distributed between 0 and a maximum value. For the root application node, the cost is divided by a factor of 10. We set the design parameter $\gamma = 2$ in the simulation.

Figures 3.5 and 3.6 show the average maximum resource utilization (averaged over results from different random seeds[8]), respectively with and without pre-specified placement of junction nodes. In Figs. 3.5(a) and 3.6(a), the number of physical nodes is randomly chosen from the interval $[2, 50]$; and in Figs. 3.5(b) and 3.6(b), the maximum resource demand (per application node/link) is set to 0.015. It is evident that the proposed method outperforms the methods in comparison. The

[7]Because the focus of our work is on MMCs, which have not been widely deployed in practice and thus realistic graphs are not available to us. Therefore, we use randomly generated graphs in our evaluation.

[8]We only consider those random seeds which produce a maximum resource utilization that is smaller than one, because otherwise, the physical network is overloaded after hosting 100 services. We also observed in the simulations that the number of accepted services is similar when using different methods. The relative degradation in the number of accepted services of the proposed method compared with other methods never exceeds 2% in the simulations.

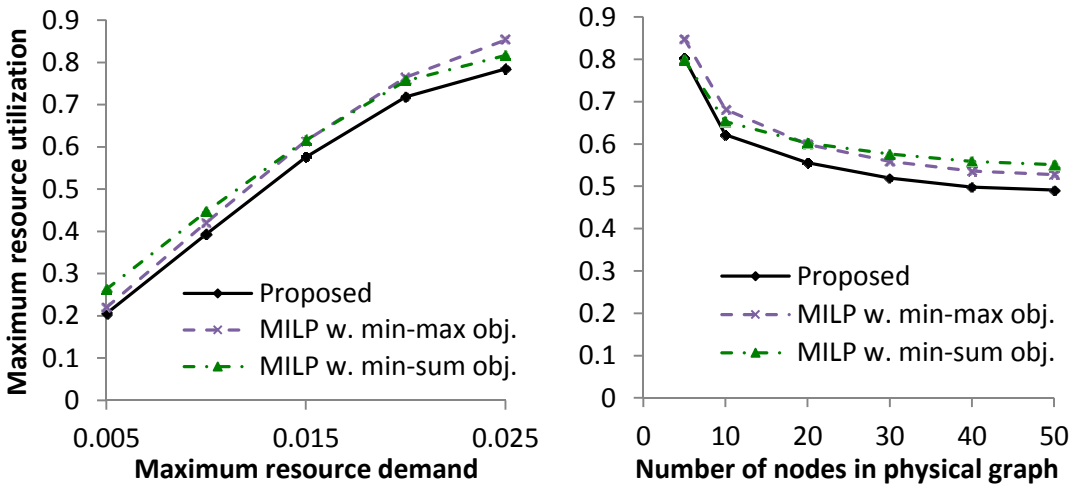


Figure 3.5: Maximum resource utilization when junction node placements are pre-specified.

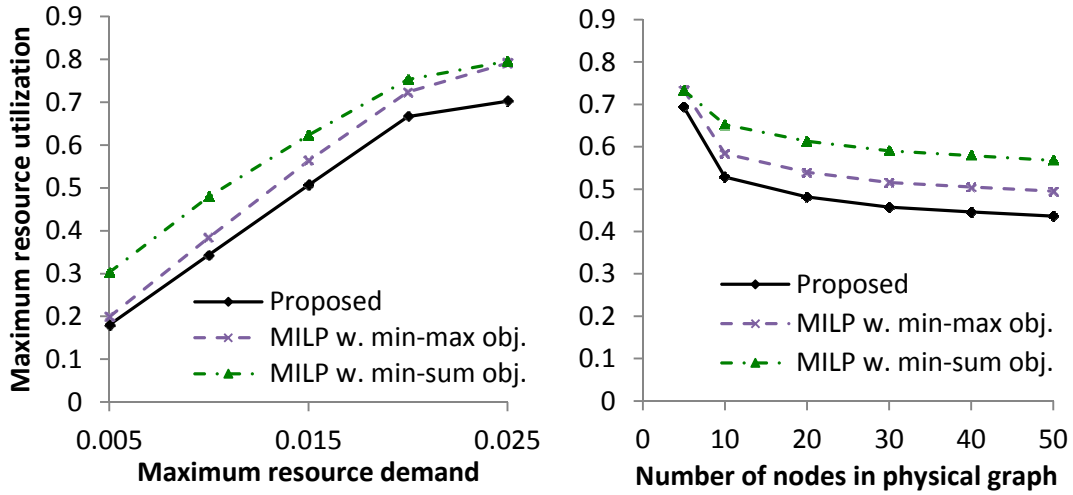


Figure 3.6: Maximum resource utilization when junction node placements are not pre-specified.

resource utilization tends to converge when the number of physical nodes is large because of the fixed root placement. As mentioned earlier, the MILP solution is the best possible solution among a set of existing approaches for online placement, which use the same objective function for each service as in the corresponding MILP formulation. Realistic online algorithms in the same set would not perform better than the MILP result, whereas solving MILP requires non-polynomial time.

The reason why the proposed method outperforms other methods is described below. We should first note that the uniqueness in the proposed algorithm is that it uses a non-linear objective function for each service, whereas the MILP methods all use linear objective functions (otherwise it is not possible to formulate with MILP, and obtaining exact solutions to non-linear programs with integer variables generally requires excessive computational time[9], thus we do not compare against exact solutions for the exponential-difference cost). The exponential-difference cost used in the proposed algorithm is indeed related to both load balancing and reducing sum resource utilization. It leaves more available resources for services that arrive in the future, compared to the MILP with min-max objective which greedily optimizes (3.3) for each new service. The MILP with min-sum objective function does not strongly enforce load balancing unless operating closely to the resource saturation point, because of the characteristics of its objective function.

When comparing Fig. 3.5 with Fig. 3.6, we can find that the performance gaps between the proposed method and other methods are larger when the junction nodes are not placed beforehand. This is mainly because the judgment of whether Algorithm 3.4 has failed is based on the factor β^{1+H}, and for Algorithm 3.2 it is based on β. It follows that Algorithm 3.4 is less likely to fail when $H > 0$. In this case, the value of $\hat{J}$ is generally set to a smaller value by the doubling procedure in Algorithm

[9]This is by noting that even linear programs with integer variables are generally NP-hard, and the best known algorithms for solving NP-hard problems have exponential time-complexity. A non-linear program is usually more difficult to solve than a linear program.

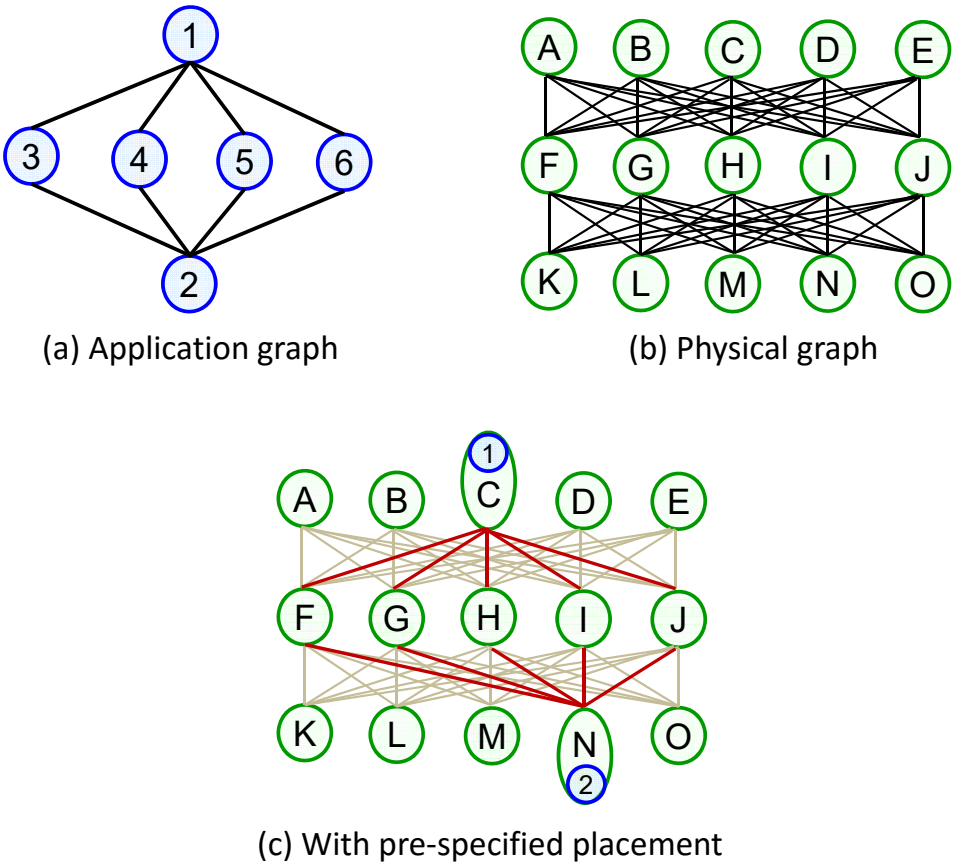
3.3. A smaller value of $\hat{J}$ also results in a larger change in the exponential-difference cost when the amount of existing load changes[10]. This brings a better load balancing on average (but not for the worst-case, the worst-case result is still bounded by the bounds derived earlier in this chapter).

3.6 Discussion

Is the Tree Assumption Needed? For ease of presentation and considering the practical relevance to MMC applications, we have focused on tree-to-tree placements in this chapter. However, the tree assumption is *not* absolutely necessary for our algorithm to be applicable. For example, consider the placement problem shown in Fig. 3.7, where the application graph consists of two junction nodes (nodes 1 and 2) and multiple simple branches (respectively including nodes 3, 4, 5, and 6) between these two junction nodes. Such an application graph is common in applications where processing can be parallelized at some stage. The physical graph shown in Fig. 3.7(b) still has a hierarchy, but we now have connections between all pairs of nodes at two adjacent levels. Obviously, neither the application nor the physical graph in this problem has a tree structure.

Let us assume that junction node 1 has to be placed at the top level of the physical graph (containing nodes A, B, C, D, E), junction node 2 has to be placed at the bottom level of the physical graph (containing nodes K, L, M, N, O), and application nodes $3, 4, 5, 6$ have to be placed at the middle level of the physical graph (containing nodes F, G, H, I, J). One possible junction node placement under this restriction is shown in Fig. 3.7(c). With this pre-specified junction node placement, the mapping of each application node in $\{3, 4, 5, 6\}$ can be found by the simple branch placement algorithm (Algorithm 3.3 which embeds Algorithm 3.2) introduced earlier, because

[10]This is except for the top-level instance of Unplaced(v, h) due to the division by β^h in Line 20 of Algorithm 3.4.



(a) Application graph

(b) Physical graph

(c) With pre-specified placement

Figure 3.7: Example where application and physical graphs are not trees: (a) application graph, (b) physical graph, (c) restricted physical graph with pre-specified placement of application nodes 1 and 2.

it only needs to map each application node in $\{3, 4, 5, 6\}$ onto each physical node in $\{$F, G, H, I, J$\}$, and find the particular assignment that minimizes (3.12) with (3.13a) and (3.13b). Therefore, in this example, when the junction node placements are pre-specified, the proposed algorithm can find the placement of other application nodes with $O(V^3N^2)$ time-complexity, which is the complexity of Algorithm 3.3 as discussed in Section 3.4.2.2. When the junction node placements are not pre-specified, the proposed algorithm can find the placement of the whole application graph with $O(V^3N^4)$ time-complexity, because here $H = 2$ (recall that the complexity result was derived in Section 3.4.3.2).

We envision that this example can be generalized to a class of application and physical graphs where there exist a limited number of junction nodes (where we can define an arbitrary node in the application graph as the root node) that are not placed beforehand. The algorithms proposed in this chapter should still be applicable to such cases, as long as we can find a limited number of cycle-free paths between two junction nodes when they are placed on the physical graph. We leave a detailed discussion to this aspect as future work.

Practical Implications: Beside the proposed algorithms themselves, the results of this chapter can also provide the following insights that may guide future implementation:

1. The placement is easier when the junction nodes are placed beforehand. This is obvious when comparing the time-complexity and competitive ratio for cases with and without unplaced junction nodes.

2. There is a trade-off between instantaneously satisfying the objective function and leaving more available resources for future services. Leaving more available resources may cause the system to operate in a sub-optimal state for the short-term, but future services may benefit from it. This trade-off can be con-

trolled by defining an alternative objective function which is different from (but related to) the overall objective that the system tries to achieve (see Section 3.4.2.1).

Comparison with [25]: As mentioned in Section 3.1, [25] is the only work which we know that has studied the competitive ratio of online service placement considering both node and link optimization. Our approach has several benefits compared to [25] as discussed in Section 3.1.2. Beside those benefits, we would like to note that the proposed algorithm outperforms [25] in time-complexity, space-complexity, and competitive ratio when the junction node placements are pre-specified (the performance bounds of these two approaches can be found in Sections 3.1.1 and 3.1.3, respectively). When some junction node placements are not pre-specified, our approach provides a performance bound comparable to that in [25], where we also note that $H \leq D$. Moreover, [25] does not consider exact optimal solutions for the placement of a single linear application graph, and it also does not have simulations to show the average performance of the algorithm.

Tightness of Competitive Ratio: By comparing the competitive ratio result of our approach to that in [25], we see that both approaches provide poly-log competitive ratios for the general case. It is however unclear whether this is the best performance bound one can achieve for the service placement problem. This is an interesting but difficult aspect worth studying in the future.

3.7 Summary

We have focused on the placement of an incoming stream of application graphs onto a physical graph in this chapter. We started with a basic assignment module for placing a single application graph that is a line, based on which we developed online

algorithms for placing tree application graphs. The performance of the proposed algorithms has been studied both analytically and via simulation.

Until now, we have proposed algorithms for initial service placement as discussed in Section 1.1.1. The next question we need to address is whether/where to migrate services in real time when mobile users move after the initial placement. This is discussed in the next two chapters.

CHAPTER 4

An MDP-Based Approach to Dynamic Service Migration

4.1 Introduction

We considered the initial placement of an arriving stream of application graphs onto the physical graph in the previous chapter. Because mobile users may move after initial placement (as discussed in Section 1.1.2) and such a movement essentially changes the physical graph topology, the initial placement may no longer be optimal or near-optimal after such a change. Thus, in this chapter, we consider the case where the cost of a particular choice of service placement varies over time, and this cost variation is mainly due to user mobility. In this case, we need to make decisions related to whether/where we should migrate the service.

As illustrated in the example given in Section 1.1.2, application components running on MMCs are the most prone to user movements, due to the relatively small coverage area of network entities (such as basestations) MMCs are directly connected to. Thus, for simplicity, in this and remaining chapters we only consider a star (two-level tree) application graph with one root node running on an MMC and one or multiple leaf nodes running on users. When the application graph only has one leaf node, it corresponds to the case where the service only serves one user. When it has multiple leaf nodes, the service serves multiple users simultaneously. We will mainly focus on the first case where each user independently runs its own

service instance on an MMC, but several approaches we present can be directly applied or extended to the second case where multiple users are served by one service instance. The service placement and migration problem with user dynamics mainly focuses on where (on which cloud) to place the root application node. We will soon see that this problem is already very complex even with these simplifications, where user dynamics is the main cause for complexity.

4.1.1 Related Work

Most existing work on service migration focuses on wide-area migrations in response to workload and energy costs that vary slowly [38, 39]. In MMCs, user mobility is the driving factor of migration, which varies much faster than parameters in conventional clouds.

The performance of MMCs with the presence of user mobility is studied in [18], but decisions on whether and where to migrate the service is not considered. A preliminary work on mobility-driven service migration based on Markov Decision Processes (MDPs) is given in [19], which mainly considers 1-D mobility patterns with a specifically defined cost function. Standard solution procedures are used to solve this MDP, which can be time-consuming especially when the MDP has a large number of states. Due to real-time dynamics, the cost functions and transition probabilities of the MDP may change rapidly over time, thus it is desirable to solve the MDP in an effective manner. Furthermore, 2-D mobility has not been considered in the literature to the best of our knowledge, which is a much more realistic case compared to 1-D mobility.

We also note that a related area of work, which is relevant to the user mobility, studies handover policies in the context of cellular networks [40]. However, the notion of service migration is very different from cellular handover. Handover is usually decided by signal strengths from different basestations, and a handover must

occur if a user's signal is no longer provided satisfactorily by its original basestation. In the service migration context, a user may continue receiving service from an MMC even if it is no longer associated with that basestation, because the user can communicate with a remote MMC via its currently associated basestation and the backhaul network. As a result, the service for a particular user can potentially be placed on any MMC, and the service migration problem has a much larger decision space than the cellular handover problem.

4.1.2 Main Results

In this chapter, we use the MDP framework to study service migration in MMCs. We provide novel contributions beyond [19], by considering more efficient solution methods, general cost models, 2-D user mobility, and application to real-world scenarios. The details are described as follows.

1. We formulate the mobility-driven dynamic service placement/migration problem with general cost models and provide a mathematical framework to design optimal service migration policies.

2. We consider 1-D user mobility[1] with a constant cost model, and propose an optimal threshold policy to solve for the optimal action of the MDP, which is more efficient than standard solution techniques. A threshold policy means that we always migrate the service for a user from one micro-cloud to another when the user is in states bounded by a particular set of thresholds, and do not migrate otherwise. We first prove the existence of an optimal threshold policy and then propose an algorithm with polynomial time-complexity for finding the optimal thresholds. The analysis with 1-D mobility model can also help

[1]The 1-D mobility is an important practical scenario often encountered in transportation networks, such as vehicles moving along a road.

us gain new insights into the migration problem, which set the foundation for more complicated scenarios studied next.

3. We consider 2-D user mobility with a more general cost model, where the cost can be expressed in a constant-plus-exponential form. For this case, we propose the following:

   (a) We note that the resulting problem becomes difficult to solve due to the large state space. In order to overcome this challenge, we propose an approximation of the underlying state space where we define the states as the *distance* between the user and the service locations[2]. This approximation becomes exact for uniform 1-D mobility. We prove several structural properties of the distance-based MDP, which includes a closed-form solution to the discounted sum cost. We leverage these properties to develop an algorithm for computing the optimal policy, which reduces the per-iteration complexity from $O(N^3)$ (by policy iteration [41, Chapter 6]) to $O(N^2)$, where N is the number of states in the distance-based MDP excluding the state corresponding to zero distance.

   (b) We show how to use the distance-based MDP to approximate the solution for 2-D mobility models, which allows us to efficiently compute a service migration policy for 2-D mobility. For uniform 2-D mobility, the approximation error is bounded by a constant. Simulation results comparing our approximation solution to the optimal solution (where the optimal solution is obtained from a 2-D MDP) suggest that it performs very close to optimal, and the proposed approximation approach obtains the solution significantly faster.

[2]Throughout this chapter, we mean by *user location* the location of the basestation that the user is associated to.

(c) We demonstrate how to apply our algorithms in a practical scenario driven by real mobility traces of taxis in San Francisco which involve multiple users and services. The practical scenario includes realistic factors, e.g., not every basestation has an MMC connected to it, and each MMC can only host a limited number of services. We compare the proposed policy with several baseline strategies that include myopic, never-migrate, and always-migrate policies. It is shown that the proposed approach offers significant gains over these baseline approaches.

4.2 Problem Formulation

Consider a mobile user that accesses a cloud-based service hosted on the MMCs. The set of possible user locations is given by $\mathcal{Q}$, where $\mathcal{Q}$ is assumed to be finite (but arbitrarily large). We consider a time-slotted model where the user's location remains fixed for the duration of one slot and changes from one slot to the next according to a Markovian mobility model. The time-slotted model can be regarded as a sampled version of a continuous-time model, and the sampling can be performed either at equal intervals over time or occur right after a cellular handover instance. In addition, we assume that each location $\varphi \in \mathcal{Q}$ is associated with an MMC that can host the service for the user. The locations in $\mathcal{Q}$ are represented as 2-D vectors and there exists a distance metric $\|\varphi_1 - \varphi_2\|$ that can be used to calculate the distance between locations φ_1 and φ_2. Note that the distance metric may not be Euclidean distance. An example of this model is a cellular network in which the user's location is taken as the location of its current basestation and the MMCs are co-located with the basestations. As shown in Section 4.4.3, these locations can be represented as 2-D vectors (i, j) with respect to a reference location (represented by $(0, 0)$) and the

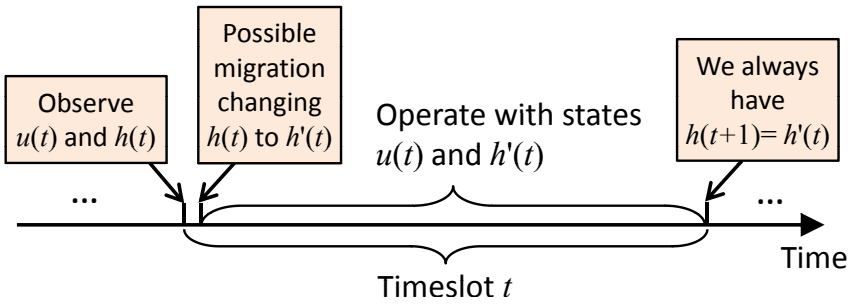


Figure 4.1: Timing of the proposed service migration mechanism.

distance between any two locations can be calculated in terms of the number of hops to reach from one cell to another cell. We denote the user and service locations at timeslot t as $u(t)$ and $h(t)$ respectively.

Remark: The problem is formulated for the case of a single user accessing a single service, i.e., the application graph only contains two nodes, one running at one of the MMCs, and the other running at the user device. However, our solution can be applied to manage services for multiple users in a heuristic manner. We will illustrate such an application in Section 4.4.4, where we also consider aspects including that MMCs are only deployed at a subset of basestations and a limited number of services can be hosted at each MMC. We assume in this chapter that different services are independent of each other, and one service serves a single user. The notion of "service" in this chapter can also stand for an instance of a particular type of service, but we do not distinguish between services and instances in this chapter for simplicity.

4.2.1 Control Decisions and Costs

At the beginning of each slot, the MMC controller can choose from one of the following control options:

1. Migrate the service from location $h(t)$ to some other location $h'(t) \in \mathcal{Q}$. This incurs a *migration cost* $c_m(x)$ that is assumed to be a non-decreasing function

of x, where x is the distance between $h(t)$ and $h'(t)$, i.e., $x = \|h(t) - h'(t)\|$. Once the migration is completed, the system operates under state $(u(t), h'(t))$. We assume that the time to perform migration is negligible compared to the time scale of user mobility (as shown in Fig. 4.1).

2. Do not migrate the service. In this case, we have $h'(t) = h(t)$ and the migration cost is $c_m(0) = 0$.

In addition to the migration cost, there is a data *transmission cost* incurred by the user for connecting to the currently active service instance. The transmission cost is related to the distance between the service and the user after possible migration, and it is defined as a general non-decreasing function $c_d(y)$, where $y = \|u(t) - h'(t)\|$. We set $c_d(0) = 0$.

4.2.2 Performance Objective

Let us denote the overall system state at the beginning of each timeslot (before possible migration) by $s(t) = (u(t), h(t))$. The state $s(t)$ is named as the *initial state* of slot t. Consider any policy π that makes control decisions based on the state $s(t)$ of the system, and we use $a_\pi(s(t))$ to represent the control action taken when the system is in state $s(t)$. This action causes the system to transition to a new *intermediate state* $s'(t) = (u(t), h'(t)) = a_\pi(s(t))$. We use $C_{a_\pi}(s(t))$ to denote the sum of migration and transmission costs incurred by a control $a_\pi(s(t))$ in slot t, and we have $C_{a_\pi}(s(t)) = c_m(\|h(t) - h'(t)\|) + c_d(\|u(t) - h'(t)\|)$. Starting from any initial state $s(0) = s_0$ (assuming that the current slot has index 0), the long-term expected *discounted sum cost* incurred by policy π is given by

$$V_\pi(s_0) = \lim_{t\to\infty} \mathbb{E}\left\{ \sum_{\tau=0}^{t} \gamma^\tau C_{a_\pi}(s(\tau)) \middle| s(0) = s_0 \right\} \tag{4.1}$$

where $0 < \gamma < 1$ is a discount factor which is related to how far we look-ahead in the cost computation. A large value of γ gives more weight to future costs, whereas a small value of γ gives less weight to future costs.

Our objective is to design a control policy that minimizes the long-term expected discounted sum total cost starting from any initial state, i.e.,

$$V^*(s_0) = \min_{\pi} V_\pi(s_0) \quad \forall s_0. \tag{4.2}$$

This problem falls within the class of MDPs with infinite horizon discounted cost. It is well known that the optimal solution is given by a stationary policy and can be obtained as the unique solution to the Bellman's equation:

$$V^*(s_0) = \min_{a}\Big\{C_a(s_0) + \gamma \sum_{s_1 \in \mathcal{Q}\times\mathcal{Q}} P_{a(s_0),s_1} V^*(s_1)\Big\} \tag{4.3}$$

where $P_{a(s_0),s_1}$ denotes the probability of transitioning from state $s'(0) = s'_0 = a(s_0)$ to $s(1) = s_1$. Note that the intermediate state $s'(t)$ has no randomness when $s(t)$ and $a(\cdot)$ are given, thus we only consider the transition probability from $s'(t)$ to the next state $s(t+1)$ in (4.3). Also note that we always have $h(t+1) = h'(t)$.

4.2.3 Characteristics of Optimal Policy

We next characterize some structural properties of the optimal solution. The following proposition states that it is not optimal to migrate the service to a location that is farther away from the user than the current service location, as one would intuitively expect.

Proposition 4.1. *Let $a^*(s) = (u, h')$ denote the optimal action at any state $s = (u, h)$. Then, we have $\|u - h'\| \leq \|u - h\|$. (If the optimal action is not unique, then there exists at least one such optimal action.)*

Corollary 4.1. *If $c_m(x)$ and $c_d(y)$ are both constants (possibly of different values) for $x, y > 0$, and $c_m(0) < c_m(x)$ and $c_d(0) < c_d(y)$ for $x, y > 0$, then migrating to locations other than the current location of the mobile user is not optimal.*

See Appendix C.2 for the proofs of Proposition 4.1 and Corollary 4.1.

4.2.4 Generic Notations

In the remainder of this chapter, where there is no ambiguity, we reuse the notations P, $C_a(\cdot)$, $V(\cdot)$, and $a(\cdot)$ to respectively represent transition probabilities, one-timeslot costs, discounted sum costs, and actions of different MDPs.

4.3 Constant Cost Model under 1-D Mobility

In this section, we consider the simple case where users follow a 1-D mobility model.

4.3.1 Definitions

We consider a 1-D region partitioned into a discrete set of areas, each of which is served by an MMC, as shown in Fig. 1.1. Such a scenario models user mobility on roads, for instance.

Mobile users are assumed to follow a 1-D asymmetric random walk mobility model. In every new timeslot, a user moves with probability p (or q) to the area that is on the right (or left) of its previous area, it stays in the same area with probability $1 - p - q$. If the system is sampled at handoff instances, then $1 - p - q = 0$, but we consider the general case with $0 \leq 1 - p - q \leq 1$. Obviously, this mobility model can be described as a Markov chain.

The state of the user is defined as the *offset* between the mobile user location and the location of the MMC running the service at the beginning of a slot, before

possible service migration, i.e., the state in slot t is defined as $e(t) = u(t) - h(t)$, where $u(t)$ is the location (index of area) of the mobile user, and $h(t)$ the location (index of area) of the MMC hosting the service. Note that $e(t)$ can be zero, positive, or negative. For simplicity, we omit the variable t and the policy subscript π in the following.

Because we consider a scenario where all MMCs are connected via the backhaul (as shown in Fig. 1.1), and the backhaul can be regarded as a central entity (which resembles the case for cellular networks, for example), in this section, we define the migration and transmission costs respectively as follows:

$$c_m(x) = \begin{cases} 0, & \text{if } x = 0 \\ 1, & \text{if } x > 0 \end{cases} \tag{4.4}$$

$$c_d(y) = \begin{cases} 0, & \text{if } y = 0 \\ \xi, & \text{if } y > 0 \end{cases}. \tag{4.5}$$

This gives the following one-timeslot cost function for taking action a in state e:

$$C_a(e) = \begin{cases} 0, & \text{if no migration or data transmission, i.e., } e = a(e) = 0 \\ \xi, & \text{if only data transmission, i.e., } e = a(e) \neq 0 \\ 1, & \text{if only migration, i.e., } e \neq a(e) = 0 \\ \xi + 1, & \text{if both migration and data transmission, i.e., } e \neq a(e) \neq 0 \end{cases} \tag{4.6}$$

Equation (4.6) is explained as follows. If the action a under state e causes no migration or data transmission (e.g., if the user and the MMC hosting the service are in the same location, i.e., $e = 0$, and we do not migrate the service to another location), we do not need to communicate via the backhaul network, and the cost is zero. A non-

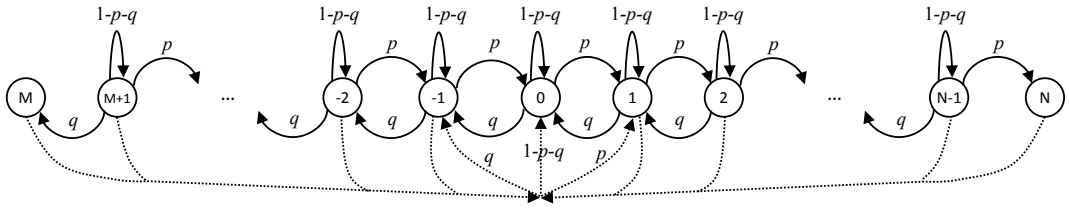


Figure 4.2: MDP model for service migration. The solid lines denote transition under action $a = 0$ and the dotted lines denote transition under action $a = 1$. When taking action $a = 1$ from any state, the next state is $e = -1$ with probability q, $e = 0$ with probability $1 - p - q$, or $e = 1$ with probability p.

zero cost is incurred when the user and the MMC hosting the service are in different locations (i.e., $u \neq h$). In this case, if we do not migrate the service to the current user location at the beginning of the timeslot (i.e., $u \neq h = h'$), the data between the MMC and mobile user need to be transmitted via the backhaul network. This data transmission incurs a cost of ξ. When migrating to the current user location (i.e., $u = h' \neq h$), we need resources to support migration, incurring a migration cost that is assumed to be 1, i.e., the cost $C_a(e)$ is normalized by the migration cost. Finally, if we migrate to a location other than the current user location, so that data transmission still occurs after migration (i.e., $u \neq h' \neq h$), the cost is $\xi + 1$.

4.3.1.1 Characteristics of the Optimal Solution

Note that the cost definitions in (4.4) and (4.5), thus (4.6), satisfy the conditions for Corollary 4.1. Therefore, we only have two candidates for optimal actions, which are migrating to the current user location and not migrating. This simplifies the action space to two actions: a migration action, denoted as $a =$ "migrate"; and a no-migration action, denoted as $a =$ "not migrate". In practice, there is usually a limit on the maximum allowable distance between the mobile user and the MMC hosting its service for the service to remain usable. We model this limitation by a maximum negative offset M and a maximum positive offset N (where $M < 0, N > 0$), such that the service must be migrated ($a =$ "migrate") when the system enters state M

or N. This implies that, although the user can move in an unbounded space, the state space of our MDP for service control is finite. The overall transition diagram of the resulting MDP is illustrated in Fig. 4.2. Note that because each state transition is the concatenated effect of (possible) migration and user movement, and the states are defined as the offset between user and service locations, the next state after taking a migration action is either -1, 0, or 1.

4.3.1.2 Modified Cost Function

With the above considerations, the cost function in (4.6) can be expressed as the following:

$$C_a(e) = \begin{cases} 0, & \text{if } e = 0 \\ \xi, & \text{if } e \neq 0, M < e < N, a = \text{“not migrate”} \\ 1, & \text{if } e \neq 0, M \leq e \leq N, a = \text{“migrate”} \end{cases} \tag{4.7}$$

4.3.1.3 Modified Bellman's Equations

With the one-timeslot cost defined as in (4.7), we obtain the following Bellman's equations for the discounted sum cost when respectively taking actions $a =$ “not migrate” and $a =$ “migrate” at state e:

$$V(e|a = \text{“not migrate”}) = \begin{cases} \gamma \sum_{j=-1}^{1} P_{0j} V(j), & \text{if } e = 0 \\ \xi + \gamma \sum_{j=e-1}^{e+1} P_{ej} V(j), & \text{if } e \neq 0, M{<}e{<}N \end{cases} \tag{4.8}$$

$$V(e|a = \text{“migrate”}) = \begin{cases} \gamma \sum_{j=-1}^{1} P_{0j} V(j), & \text{if } e = 0 \\ 1 + \gamma \sum_{j=-1}^{1} P_{0j} V(j), & \text{if } e \neq 0, M{\leq}e{\leq}N \end{cases} \tag{4.9}$$

where the transition probability P_{ij} is related to parameters p and q as defined earlier. The discounted sum cost $V(e)$ when following the optimal policy is

$$V(e) = \begin{cases} \min\{V(e|a = \text{“not migrate”}), V(e|a = \text{“migrate”})\}, & \text{if } M < e < N \\ V(e|a = \text{“migrate”}), & \text{if } e = M \text{ or } e = N \end{cases} \tag{4.10}$$

4.3.2 Optimal Threshold Policy

4.3.2.1 Existence of Optimal Threshold Policy

We first show that there exists a threshold policy which is optimal for the MDP in Fig. 4.2.

Proposition 4.2. *There exists a threshold policy (k_1, k_2), where $M < k_1 \leq 0$ and $0 \leq k_2 < N$, such that when $k_1 \leq e \leq k_2$, the optimal action for state e is $a^*(e) =$ “not migrate”, and when $e < k_1$ or $e > k_2$, $a^*(e) =$ “migrate”.*

Proof. It is obvious that different actions for state zero $a(0) =$ “not migrate” and $a(0) =$ “migrate” are essentially the same, because the mobile user and the MMC hosting its service are in the same location under state zero, either action does not incur cost and we always have $C_{a(0)}(0) = 0$. Therefore, we can conveniently choose $a^*(0) =$ “not migrate”.

In the following, we show that, if it is optimal to migrate at $e = k_1 - 1$ and $e = k_2 + 1$, then it is optimal to migrate at all states e with $M \leq e \leq k_1 - 1$ or $k_2 + 1 \leq e \leq N$. We relax the restriction that we always migrate at states M and N for now, and later discuss that the results also hold for the unrelaxed case. We only focus on $k_2 + 1 \leq e \leq N$, because the case for $M \leq e \leq k_1 - 1$ is similar.

If it is optimal to migrate at $e = k_2 + 1$, we have

$$V(k_2 + 1|a = \text{“migrate”}) \leq \xi \sum_{t=0}^{\infty} \gamma^t = \frac{\xi}{1-\gamma} \tag{4.11}$$

where the right hand-side of (4.11) is the discounted sum cost of a never-migrate policy supposing that the user never returns back to state zero when starting from state $e = k_2 + 1$. This cost is an upper bound of the cost incurred from any possible state-transition path without migration, and migration cannot bring higher cost than this because otherwise it contradicts the presumption that it is optimal to migrate at $e = k_2 + 1$.

Suppose we do not migrate at a state e' where $k_2 + 1 < e' \leq N$, then we have a (one-timeslot) cost of ξ in each timeslot until the user reaches a migration state (i.e., a state at which we perform migration). From (4.9), we know that $V(e|a = \text{“migrate”})$ is constant for any $e \neq 0$. Therefore, any state-transition path L starting from state e' has a discounted sum cost of

$$V_L(e') = \xi \sum_{t=0}^{t_m-1} \gamma^t + \gamma^{t_m} V(k_2 + 1|a = \text{“migrate”})$$

where $t_m > 0$ is a parameter representing the first timeslot that the user is in a migration state after reaching state e' (assuming that we reach state e' at $t = 0$), which is dependent on the state-transition path L. We have

$$\begin{aligned}
&V_L(e') - V(k_2 + 1|a = \text{“migrate”}) \\
&= \xi \frac{(1-\gamma^{t_m})}{1-\gamma} - \left(1-\gamma^{t_m}\right) V(k_2 + 1|a = \text{“migrate”}) \\
&= \left(1-\gamma^{t_m}\right) \left(\frac{\xi}{1-\gamma} - V(k_2 + 1|a = \text{“migrate”}) \right) \geq 0
\end{aligned}$$

where the last inequality follows from (4.11). It follows that, for any possible state-transition path L, $V_L(e') \geq V(k_2 + 1|a = \text{“migrate”})$. Hence, it is always optimal

to migrate at state e', which brings cost $V(e'|a = \text{“migrate”}) = V(k_2 + 1|a = \text{“migrate”})$.

The result also holds with the restriction that we always migrate at states M and N, because no matter what thresholds (k_1, k_2) we have for the relaxed problem, migrating at states M and N always yield a threshold policy. □

Proposition 4.2 shows the existence of an optimal threshold policy. The optimal threshold policy exists for arbitrary values of M, N, p, and q.

4.3.3 Simplifying the Cost Calculation

The existence of the optimal threshold policy allows us simplify the cost calculation, which helps us develop an algorithm that has lower complexity than standard MDP solution algorithms. For the policy specified by the thresholds (k_1, k_2), the value updating function (4.10) can be changed to the following:

$$V(e) = \begin{cases} V(e|a = \text{“not migrate”}), & \text{if } k_1 \le e \le k_2 \\ V(e|a = \text{“migrate”}), & \text{otherwise} \end{cases} \tag{4.12}$$

From (4.8) and (4.9), we know that, for a given policy with thresholds (k_1, k_2), we only need to compute $V(e)$ with $k_1 - 1 \le e \le k_2 + 1$, because the values of $V(e)$ with $e \le k_1 - 1$ are identical, and the values of $V(e)$ with $e \ge k_2 - 1$ are also identical. Note that we always have $k_1 - 1 \ge M$ and $k_2 + 1 \le N$, because $k_1 > M$ and $k_2 < N$ as we always migrate when at states M and N.

Define

$$\mathbf{v}_{(k_1,k_2)} = [V(k_1 - 1)\ V(k_1) \cdots V(0) \cdots V(k_2)\ V(k_2 + 1)]^{\mathrm{T}} \tag{4.13}$$

$$\mathbf{c}_{(k_1,k_2)} = \Big[\ 1 \ \underbrace{\xi \ \cdots \ \xi}_{-k_1 \text{ elements}} \ 0 \ \underbrace{\xi \ \cdots \ \xi}_{k_2 \text{ elements}} \ 1 \ \Big]^{\mathrm{T}} \tag{4.14}$$

$$\mathbf{P}'_{(k_1,k_2)} = \begin{bmatrix} P_{0,k_1-1} & \cdots & P_{00} & \cdots & P_{0,k_2+1} \\ P_{k_1,k_1-1} & \cdots & P_{k_1,0} & \cdots & P_{k_1,k_2+1} \\ \vdots & & \vdots & & \vdots \\ P_{0,k_1-1} & \cdots & P_{00} & \cdots & P_{0,k_2+1} \\ \vdots & & \vdots & & \vdots \\ P_{k_2,k_1-1} & \cdots & P_{k_2,0} & \cdots & P_{k_2,k_2+1} \\ P_{0,k_1-1} & \cdots & P_{00} & \cdots & P_{0,k_2+1} \end{bmatrix} \tag{4.15}$$

where superscript "T" denotes the transpose of the matrix.

Then, (4.8) and (4.9) can be rewritten as

$$\mathbf{v}_{(k_1,k_2)} = \mathbf{c}_{(k_1,k_2)} + \gamma \mathbf{P}'_{(k_1,k_2)} \mathbf{v}_{(k_1,k_2)} \tag{4.16}$$

The value vector $\mathbf{v}_{(k_1,k_2)}$ can be obtained by

$$\mathbf{v}_{(k_1,k_2)} = \left(\mathbf{I} - \gamma \mathbf{P}'_{(k_1,k_2)}\right)^{-1} \mathbf{c}_{(k_1,k_2)} \tag{4.17}$$

The matrix $\left(\mathbf{I} - \gamma \mathbf{P}'_{(k_1,k_2)}\right)$ is invertible for $0 < \gamma < 1$, because in this case there exists a unique solution for $\mathbf{v}_{(k_1,k_2)}$ from (4.16), which is a known property for MDPs. Equation (4.17) can be computed using Gaussian elimination that has a complexity of $O\left((|M| + N)^3\right)$.

4.3.4 Algorithm for Finding the Optimal Thresholds

To find the optimal thresholds, we can perform a search on values of (k_1, k_2). Further, because an increase/decrease in $V(e)$ for some e increases/decreases each ele-

ment in the cost vector $\mathbf{v}$ due to cost propagation following balance equations (4.8) and (4.9), we only need to minimize $V(e)$ while considering a specific state e. We propose an algorithm for finding the optimal thresholds, as shown in Algorithm 4.1, which is a modified version of the standard[3] policy iteration mechanism [41, Chapter 6].

Algorithm 4.1 is explained as follows. We keep iterating until the thresholds no longer change, which implies that the optimal thresholds have been found. The thresholds (k_1^*, k_2^*) are those obtained from each iteration.

Lines 4–6 compute $V(e)$ for all e under the given thresholds (k_1^*, k_2^*). Then, Lines 8–22 determine the search direction for k_1 and k_2. Because $V(e)$ in each iteration is computed using the current thresholds (k_1^*, k_2^*), we have actions $a(k_1^*) = a(k_2^*) =$ "not migrate", and (4.8) is automatically satisfied when replacing its left hand-side with $V(k_1^*)$ or $V(k_2^*)$. Lines 9 and 16 check whether iterating according to (4.9) can yield lower cost. If it does, it means that migrating is a better action at state k_1^* (or k_2^*), which also implies that we should migrate at states e with $M \leq e \leq k_1^*$ (or $k_2^* \leq e \leq N$) according to Proposition 4.2. In this case, k_1^* (or k_2^*) should be set closer to zero, and we search through those thresholds that are closer to zero than k_1^* (or k_2^*). If Line 9 (or Line 16) is not satisfied, according to Proposition 4.2, it is good not to migrate at states e with $k_1^* \leq e \leq 0$ (or $0 \leq e \leq k_2^*$), so we search k_1 (or k_2) to the direction approaching M (or N), to see whether it is good not to migrate under those states.

Lines 23–37 adjust the thresholds. If we are searching toward state M or N and Line 25 is satisfied, it means that it is better not to migrate under this state (k_i), and we update the threshold to k_i. When Line 27 is satisfied, it means that it is better to migrate at state k_i. According to Proposition 4.2, we should also migrate at any state

[3]We use the term "standard" here to distinguish with the modified policy iteration mechanism proposed in Algorithm 4.1.

Algorithm 4.1 Modified policy iteration algorithm for finding the optimal thresholds

Initialize $k_1^* \leftarrow 0$, $k_2^* \leftarrow 0$
repeat
  $k_1'^* \leftarrow k_1^*$, $k_2'^* \leftarrow k_2^*$ //record previous thresholds
  Construct $\mathbf{c}_{(k_1^*,k_2^*)}$ and $\mathbf{P}'_{(k_1^*,k_2^*)}$ according to (4.14) and (4.15)
  Evaluate $\mathbf{v}_{(k_1^*,k_2^*)}$ from (4.17)
  Extend $\mathbf{v}_{(k_1^*,k_2^*)}$ to obtain $V(e)$ for all $M \leq e \leq N$
  for $i = 1, 2$ **do**
    if $i = 1$ **then**
      if $1 + \gamma \sum_{j=-1}^{1} P_{0j} V(j) < V(k_1^*)$ **then**
        dir $\leftarrow 1$, loopVec $\leftarrow [k_1^* + 1, k_1^* + 2, ..., 0]$
        $k_1^* \leftarrow k_1^* + 1$
      else
        dir $\leftarrow 0$, loopVec $\leftarrow [k_1^* - 1, k_1^* - 2, ..., M + 1]$
      end if
    else if $i = 2$ **then**
      if $1 + \gamma \sum_{j=-1}^{1} P_{0j} V(j) < V(k_2^*)$ **then**
        dir $\leftarrow 1$, loopVec $\leftarrow [k_2^* - 1, k_2^* - 2, ..., 0]$
        $k_2^* \leftarrow k_2^* - 1$
      else
        dir $\leftarrow 0$, loopVec $\leftarrow [k_2^* + 1, k_2^* + 2, ..., N - 1]$
      end if
    end if
    for $k_i =$ each value in loopVec **do**
      if dir $= 0$ **then**
        if $\xi + \gamma \sum_{j=k_i-1}^{k_i+1} P_{k_i,j} V(j) < V(k_i)$ **then**
          $k_i^* \leftarrow k_i$
        else if $\xi + \gamma \sum_{j=k_i-1}^{k_i+1} P_{k_i,j} V(j) > V(k_i)$ **then**
          exit for
        end if
      else if dir $= 1$ **then**
        if $1 + \gamma \sum_{j=-1}^{1} P_{0j} V(j) < V(k_i)$ **then**
          $k_i^* \leftarrow k_i - \text{sign}(k_i)$
        else if $1 + \gamma \sum_{j=-1}^{1} P_{0j} V(j) > V(k_i)$ **then**
          exit for
        end if
      end if
    end for
  end for
until $k_1^* = k_1'^*$ and $k_2^* = k_2'^*$
return k_1^*, k_2^*

closer to M or N than state k_i, thus we exit the loop. If we are searching toward state zero and Line 31 is satisfied, it is good to migrate under this state (k_i), therefore the threshold is set to one state closer to zero ($k_i - \text{sign}(k_i)$). When Line 33 is satisfied, we should not migrate at state k_i. According to Proposition 4.2, we should also not migrate at any state closer to zero than state k_i, and we exit the loop.

Proposition 4.3. *The threshold-pair (k_1^*, k_2^*) is different in every iteration of the loop starting at Line 2, otherwise the loop terminates.*

Proof. The loop starting at Line 2 changes k_1^* and k_2^* in every iteration so that $V(e)$ for all e become smaller. It is therefore impossible that k_1^* and k_2^* are the same as in one of the previous iterations and at the same time reduce the value of $V(e)$, because $V(e)$ computed from (4.17) is the stationary cost value for thresholds (k_1^*, k_2^*). The only case when (k_1^*, k_2^*) are the same as in the previous iteration (which does not change $V(e)$) terminates the loop. □

Corollary 4.2. *The number of iterations in Algorithm 4.1 is $O(|M|N)$.*

Proof. According to Proposition 4.3, there can be at most $|M|N + 1$ iterations in the loop starting at Line 2. □

If we use Gaussian elimination (with complexity $O\left((|M| + N)^3\right)$) to compute (4.17), the overall time-complexity of Algorithm 4.1 is $O\left(|M|N\left(|M| + N\right)^3\right)$. This is a promising result because the overall complexity of standard value or policy iteration methods for solving MDPs are generally dependent on the discount factor γ [42].

Regarding the space-complexity issue, i.e., necessary memory storage, we can see that the non-constant size arrays in Algorithm 4.1 include $\mathbf{v}_{\left(k_1^*,k_2^*\right)}$, $\mathbf{c}_{\left(k_1^*,k_2^*\right)}$, and $\mathbf{P}'_{\left(k_1^*,k_2^*\right)}$, so the space complexity of this algorithm is $O\left((|M| + N)^2\right)$, where the square is due to matrix $\mathbf{P}'$. Note that we also need to store non-truncated versions of $\mathbf{v}$, $\mathbf{c}$, and $\mathbf{P}'$, but the order of complexity remains the same. This space-complexity

is also in the same order as standard value and policy iteration algorithms which also need to store non-truncated arrays of costs and transition probabilities. We do not analyze the detailed change in memory occupation of different approaches because it can largely depend on how well the code is optimized for memory.

4.3.5 Simulation Results

In the simulations, we set the number of states $|M| = N = 10$. The transition probabilities p and q are randomly generated. Simulations are run with 1000 different random seeds in each setting to obtain the overall performance.

Distribution of Optimal Thresholds: We first study the distribution of optimal thresholds found from the proposed threshold-based method under different values of ξ. Recall that as defined in (4.6), ξ stands for the data transmission cost, while the migration cost is defined to be always equal to 1. A larger value of ξ stands for a larger transmission cost. The frequency of different optimal threshold values is shown in Fig. 4.3, where the frequency is defined as the percentage that each threshold value is found as optimum, among all 1000 simulation instances. Note that the thresholds k_1 and k_2 can only take integer values within the interval of $[-10, 10]$, but we have connected these discrete points in the plot to illustrate the overall distribution.

Several interesting observations can be seen in Fig. 4.3. First, because we always migrate at states $M = -10$ and $N = 10$ as discussed in Section 4.3.1.1, we always have $k_1^* \geq -9$ and $k_2^* \leq 9$. Thus the frequencies of thresholds $k_1 = -10$ and $k_2 = 10$ are always zero. Next, when $\xi = 0$, we always have the optimal thresholds $k_1^* = -9$ and $k_2^* = 9$, which means that we never migrate except at states M and N. We refer to such a policy as "never-migrate" in later discussion. The reason for having such optimal thresholds is because the transmission cost is zero in this case,

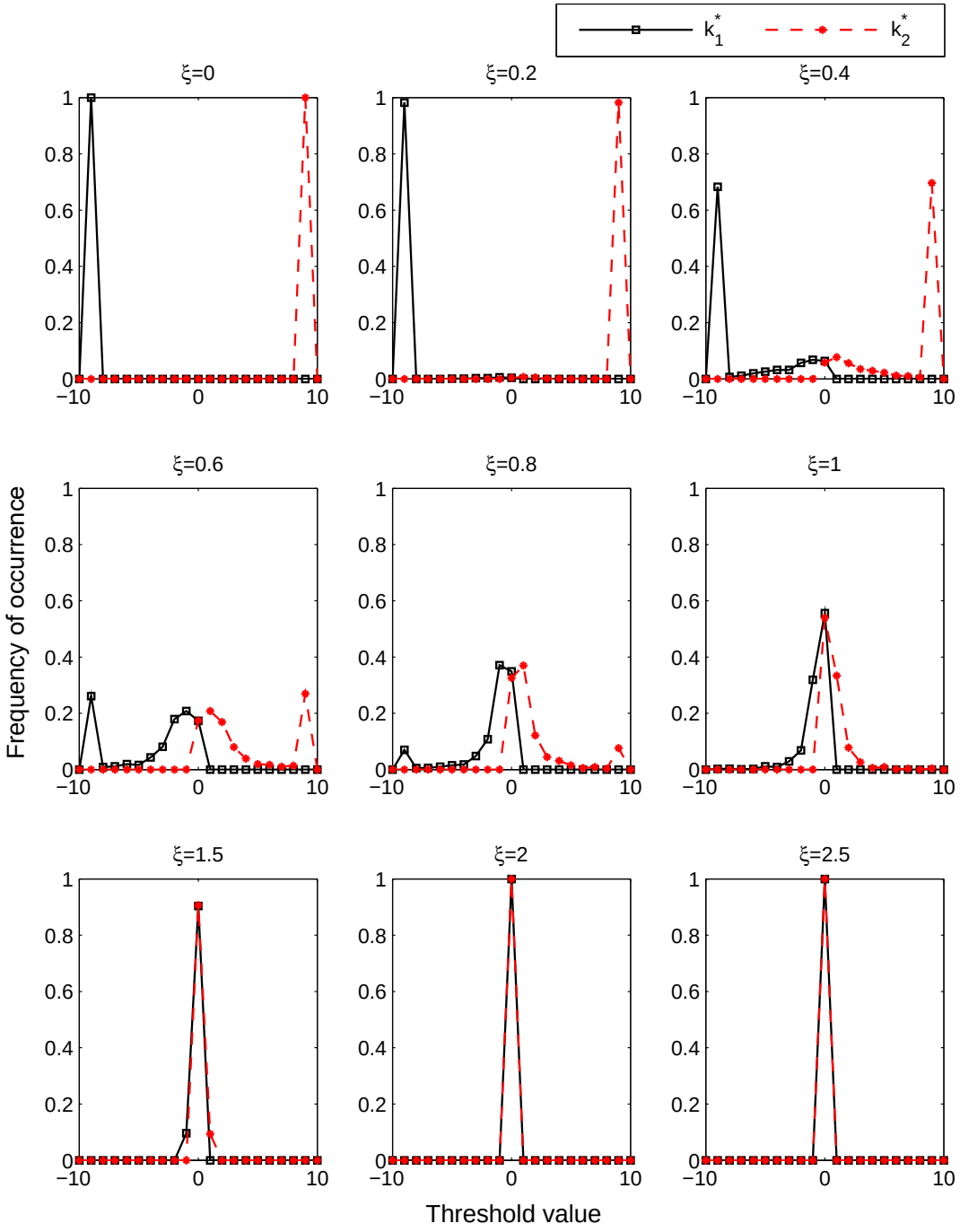


Figure 4.3: Frequency of different optimal threshold values under $\gamma = 0.9$.

and there is no benefit from performing migration, except at states M and N which is enforced in the problem formulation. When ξ increases, the optimal thresholds tend to become closer to zero, because migration is more beneficial for reducing the overall cost when the transmission cost is large. We see a spike in the frequencies for thresholds $k_1 = -9$ and $k_2 = 9$ when $\xi \leq 0.8$, due to the truncation effect of states $|M|$ and N. The simulation instances that have these optimal threshold values correspond to those that would have optimal thresholds at or beyond the $[-9, 9]$ limit if we would have larger values of $|M|$ and N. When $\xi \geq 2$, the optimal thresholds k_1^* and k_2^* are always zero, because the transmission cost is so large in this case so that it is beneficial to always migrate to the user location. We refer to such a migration policy as "always-migrate" in later discussion.

Time-Complexity and Discounted Sum Costs: We now study the time-complexity of different MDP solution approaches and discounted sum costs of different migration policies. We use two measures of time-complexity. One is the physical time of running the algorithms in MATLAB (referred to as computation time), on a computer with 64-bit Windows 7, Intel Core i7-2600 CPU, and 8GB memory, without any other foreground tasks running simultaneously (but there might be some background tasks running which are hard to control). The other measure is the number of floating-point operations (FLOPs), which is independent of background tasks thus more objective, but less intuitive than the computation time. The number of FLOPs has been collected using the toolbox in [43]. Figs. 4.4–4.6. show the computation time, number of FLOPs, and discounted sum costs under different values of ξ, respectively with $\gamma = 0.5, 0.9, 0.99$. For time complexity evaluation, we compare the proposed threshold approach against standard value and policy iteration methods [41, Chapter 6]. The value iteration terminates according to an error bound of $\varepsilon = 0.1$ in the discounted sum cost. Note

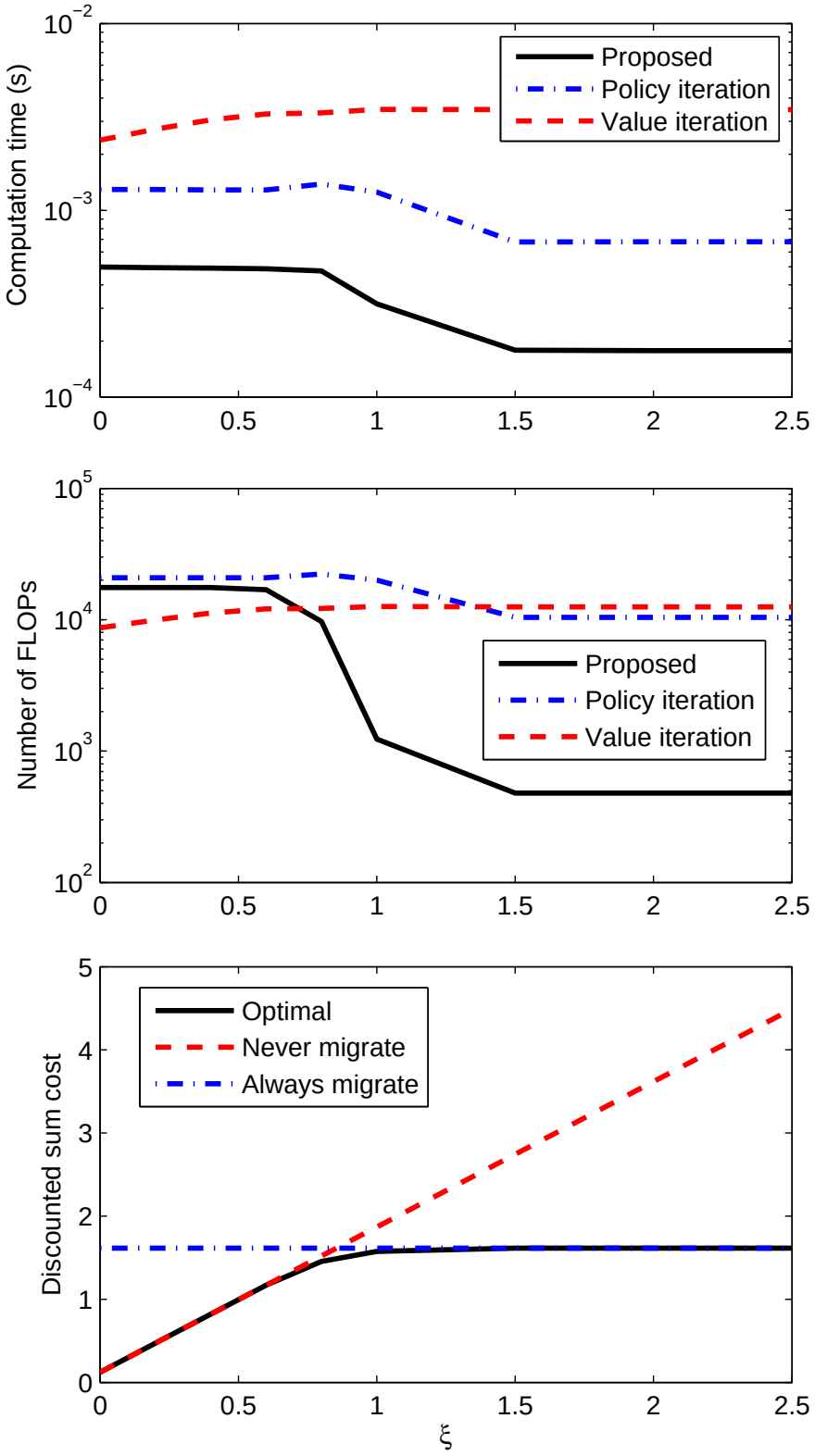


Figure 4.4: Performance under different ξ with $\gamma = 0.5$.

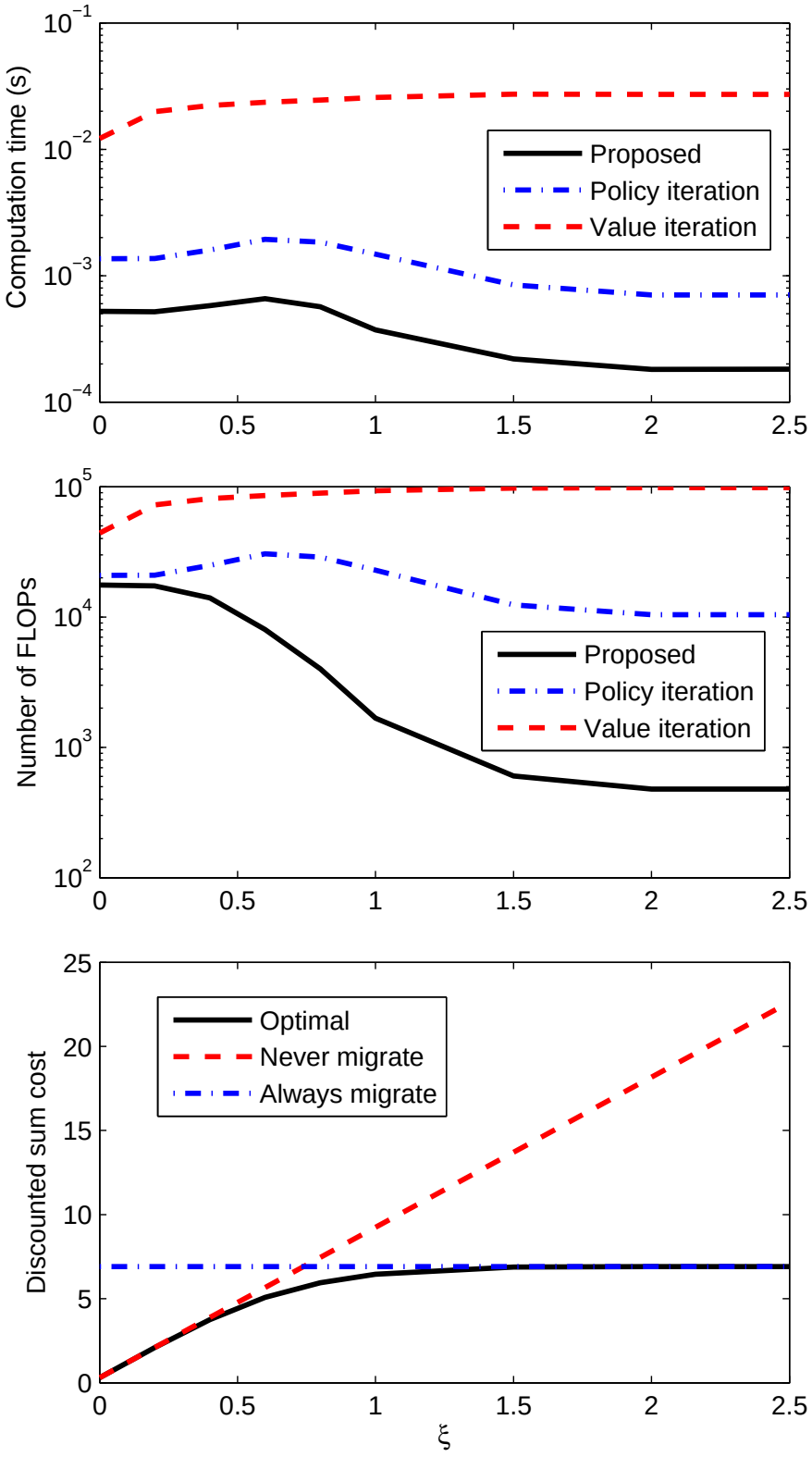


Figure 4.5: Performance under different ξ with $\gamma = 0.9$.

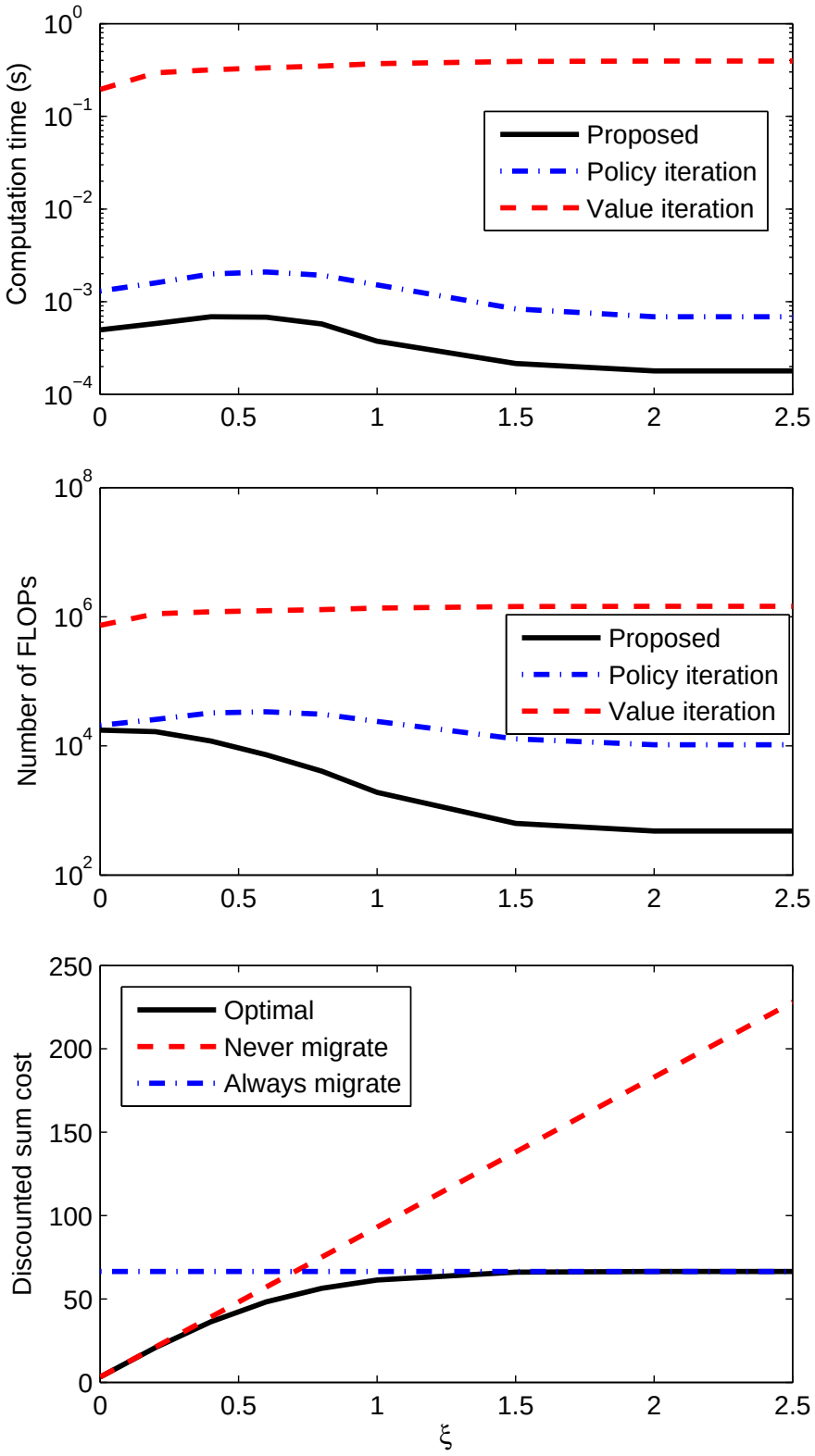


Figure 4.6: Performance under different ξ with $\gamma = 0.99$.

that the proposed method and the standard policy iteration method always incur the optimal cost. For discounted sum cost evaluation, we compare the cost of the optimal migration policy to that of never-migrate and always-migrate policies.

The results show that the proposed method always has lowest computation time and almost always has lowest number of FLOPs. This is because the proposed algorithm simplifies the solution search procedure compared to standard mechanisms. Specifically, the computation time of the standard policy iteration method is 2 to 5 times larger than that of the proposed algorithm, while the value iteration approach consumes longer time.

Compared to the computation time, the trend of the number of FLOPs varies more with different values of ξ. Particularly for the proposed approach, the number of FLOPs is high when ξ is small, and it decreases substantially when ξ is large. We think this is because the complexity of matrix inversion is adequately weighted when counting the number of FLOPs, but less weighted when measuring the computation time, because MATLAB generally computes matrix operations significantly faster than other procedures written in MATLAB code. When ξ is small, the optimal policy becomes close to a never-migrate policy (see earlier discussion on Fig. 4.3). In this case, the reduced transition probability matrix $\mathbf{P}'_{(k_1,k_2)}$ defined in (4.15) is usually large, thus a large number of FLOPs is needed to compute the inverse in (4.17). We also note here that the initial policies in all three algorithms are set as the always-migrate policy. Conversely, when ξ is large, the optimal policy is close to an always-migrate policy and the size of $\mathbf{P}'_{(k_1,k_2)}$ is usually small, resulting in a small number of FLOPs required for matrix inversion. We can also see a similar but much less obvious trend for the standard policy iteration approach, which we think is also due to the complexity of matrix inversion, because MATLAB may have mechanisms to simplify the inversion procedure according to the matrix structure.

When comparing the figures for different values of γ, we can see a clear increase

in the complexity of value iteration when γ increases. This is because for a larger γ, it takes a longer time for the discounted sum cost to converge when iterating according to (4.8)–(4.10), and value iteration is based on such iterations.

The third subfigures in each of Figs. 4.4–4.6 show the optimal cost compared to a never-migrate or always-migrate policy, where we note that the optimal cost can be found from the proposed threshold-based algorithm. As discussed earlier, we also see here that the optimal cost approaches the cost of a never-migrate policy when ξ is small, and it approaches the cost of an always-migrate policy when ξ is large. There is an intersection point of the costs of never-migrate and always-migrate policies. When $\gamma = 0.5$, the intersection point is at around $\xi = 0.9$, and it is close to the cost of optimal policy under the same value of ξ. We can infer that the optimal policy approaches either a never-migrate policy or an always-migrate policy in this case. When γ is larger, the gap between the intersection point and optimal cost is also larger. In this case, the optimal policy is likely to be neither never-migrate nor always-migrate, meaning that the optimal thresholds lie somewhere between 0 and ± 9, as shown in Fig. 4.3 when ξ takes a proper value. The reason for this phenomenon is that with a small γ, the future costs have low weights in the discounted sum cost expression (4.1), so migration decision is made mainly based on the instantaneous cost. In the extreme case where $\gamma = 0$, the optimal policy would only minimize the one-slot cost $C_a(e)$ defined in (4.7), which means it would never migrate if $\xi < 1$ and always migrate if $\xi > 1$. Therefore, the gap between the intersection point and optimal cost is small when γ is small, and large when γ is large.

4.4 Constant-Plus-Exponential Cost Model under 2-D Mobility

In this section, we consider 2-D mobility models with more general cost functions. We first outline the necessity of simplifying the search space, then introduce the distance-based MDP, which will be used to approximate the 2-D MDP later in this section.

4.4.1 Simplifying the Search Space

Because the optimality of threshold policy discussed in Section 4.3 does not directly apply to the more general case involving 2-D mobility and non-constant transmission/migration costs, we need to find alternative ways to simplify the solution space. Note that Proposition 4.1 simplifies the search space for the optimal policy considerably. However, it is still very challenging to derive the optimal control policy for the general model presented in Section 4.2, particularly when the state space $\{s(t)\}$ is large. One possible approach to address this challenge is to re-define the state space to represent only the *distance* between the user and service locations $d(t) = \|u(t) - h(t)\|$. The motivation for this comes from the observation that the cost functions in our model depend only on the distance (see cost definition in Section 4.2.1). Note that in general, the optimal control actions can be different for two states s_0 and s_1 that have the same user-service distance. However, it is reasonable to use the distance as an approximation of the state space for many practical scenarios of interest, and this simplification allows us to formulate a far more tractable MDP. We discuss the distance-based MDP in the next section, and show how the results on the distance-based MDP can be applied to 2-D mobility models and real-world mobility traces in Sections 4.4.3 and 4.4.4.

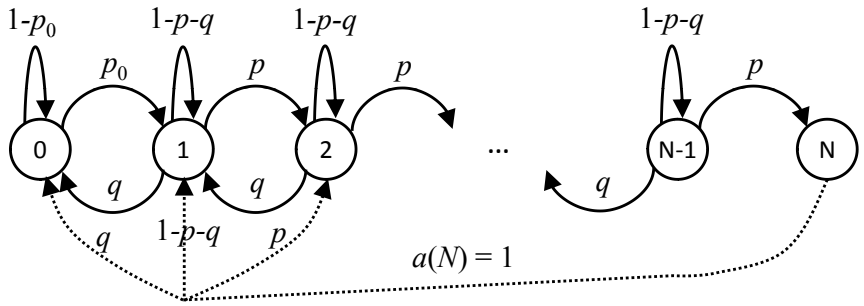


Figure 4.7: An example of distance-based MDP with the distances $\{d(t)\}$ (before possible migration) as states. In this example, migration is only performed at state N, and only the possible action of $a(N) = 1$ is shown for compactness. The solid lines denote state transitions without migration.

4.4.2 Optimal Policy for Distance-Based MDP

In this section, we consider a distance-based[4] MDP where the states $\{d(t)\}$ represent the distances between the user and the service before possible migration (an example is shown in Fig. 4.7), i.e., $d(t) = \|u(t) - h(t)\|$. Similar to Section 4.3, we define the parameter N as an application-specific maximum allowed distance, and we always perform migration when $d(t) \geq N$. We set the actions $a(d(t)) = a(N)$ for $d(t) > N$, so that we only need to focus on the states $d(t) \in [0, N]$. After taking action $a(d(t))$, the system operates in the intermediate state $d'(t) = a(d(t))$, and the value of the next state $d(t+1)$ follows the transition probability $P_{d'(t),d(t+1)}$ which is related to the mobility model of the user. To simplify the solution, we restrict the transition probabilities $P_{d'(t),d(t+1)}$ according to the parameters p_0, p, and q as shown in Fig. 4.7. Such a restriction is sufficient when the underlying mobility model is a uniform 1-D random walk where the user moves one step to the left or right with equal probability r_1 and stays in the same location with probability $1 - 2r_1$, in which case we can set $p = q = r_1$ and $p_0 = 2r_1$. This model is also sufficient to approximate the uniform 2-D random walk model, as will be discussed in Section 4.4.3.2.

For an action of $d'(t) = a(d(t))$, the new service location $h'(t)$ is chosen such

[4]We assume that the distance is quantized, as it will be the case with the 2-D model discussed in later sections.

that $x = \|h(t) - h'(t)\| = |d(t) - d'(t)|$ and $y = \|u(t) - h'(t)\| = d'(t)$. This means that migration happens along the shortest path that connects $u(t)$ and $h(t)$, and $h'(t)$ is on this shortest path (also note that $d'(t) \leq d(t)$ according to Proposition 4.1). Such a migration is possible for the 1-D case where $u(t)$, $h(t)$, and $h'(t)$ are all scalar values. It is also possible for the 2-D case if the distance metric is properly defined (see Section 4.4.3.2 for details). The one-timeslot cost is then $C_a(d(t)) = c_m(|d(t) - d'(t)|) + c_d(d'(t))$.

4.4.2.1 Constant-Plus-Exponential Cost Functions

To simplify the analysis later, we define the cost functions $c_m(x)$ and $c_d(y)$ in a constant-plus-exponential form:

$$c_m(x) = \begin{cases} 0, & \text{if } x = 0 \\ \beta_c + \beta_l \mu^x, & \text{if } x > 0 \end{cases} \tag{4.18}$$

$$c_d(y) = \begin{cases} 0, & \text{if } y = 0 \\ \delta_c + \delta_l \theta^y, & \text{if } y > 0 \end{cases} \tag{4.19}$$

where β_c, β_l, δ_c, δ_l, μ, and θ are real-valued parameters.

The functions $c_m(x)$ and $c_d(y)$ defined above can have different shapes and are thus applicable to many realistic scenarios (see Fig. 4.8 for an example). They can approximate an arbitrary cost function as discussed in Appendix B. They also have nice properties allowing us to obtain a closed-form solution to the discounted sum cost, based on which we design an efficient algorithm for finding the optimal policy.

The parameters β_c, β_l, δ_c, δ_l, μ, and θ are selected such that $c_m(x) \geq 0$, $c_d(y) \geq 0$, and both $c_m(x)$ and $c_d(y)$ are non-decreasing respectively in x and y for $x, y \geq 0$. Explicitly, we have $\mu \geq 0$; $\beta_l \leq 0$ when $\mu \leq 1$; $\beta_l \geq 0$ when $\mu \geq 1$; $\beta_c \geq -\beta_l$; $\theta \geq 0$; $\delta_l \leq 0$ when $\theta \leq 1$; $\delta_l \geq 0$ when $\theta \geq 1$; and $\delta_c \geq -\delta_l$. The definition that

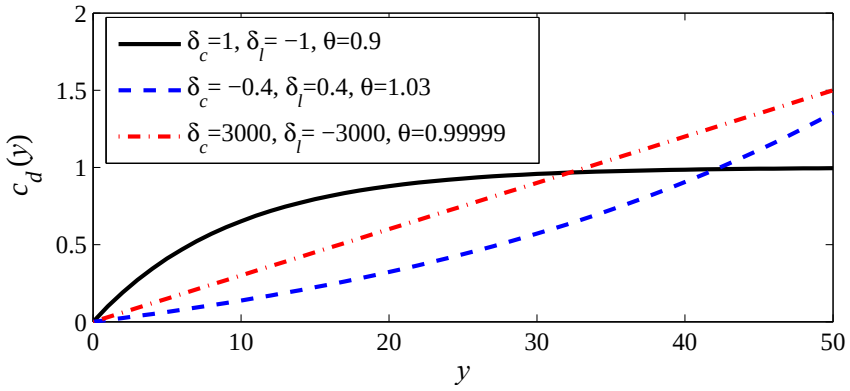


Figure 4.8: Example of constant-plus-exponential cost function $c_d(y)$.

$c_m(0) = c_d(0) = 0$ is for convenience, because a non-zero cost for $x = 0$ or $y = 0$ can be offset by the values of β_c and δ_c, thus setting $c_m(0) = c_d(0) = 0$ does not affect the optimal decision.

With this definition, the values of $\beta_c + \beta_l$ and $\delta_c + \delta_l$ can be regarded as constant terms of the costs, at least such an amount of cost is incurred when $x > 0$ and $y > 0$, respectively. The parameters μ and θ specify the impact of the distance x and y, respectively, to the costs, and their values can be related to the network topology and routing mechanism of the network. The parameters β_l and δ_l further adjust the costs proportionally.

4.4.2.2 Closed-Form Solution to Discounted Sum Cost

Problem Formulation with Difference Equations: From (4.1), we can get the following balance equation on the discounted sum cost for a given policy π:

$$V_\pi(d(0)) = C_{a_\pi}(d(0)) + \gamma \sum_{d(1)=a_\pi(d(0))-1}^{a_\pi(d(0))+1} P_{a_\pi(d(0)),d(1)} V_\pi(d(1)). \tag{4.20}$$

In the following, we will omit the subscript π and write $d(0)$ as d for short.

Proposition 4.4. *For a given policy π, let $\{\tilde{n}_k : k \geq 0\}$ denote the series of all migration states (such that $a(\tilde{n}_k) \neq \tilde{n}_k$) as specified by policy π, where $0 \leq \tilde{n}_k \leq N$.*

The discounted sum cost $V(d)$ for $d \in [\tilde{n}_{k-1}, \tilde{n}_k]$ (where we define $\tilde{n}_{-1} \triangleq 0$ for convenience) when following policy π can be expressed as

$$V(d) = A_k \zeta_1^d + B_k \zeta_2^d + D + \begin{cases} H \cdot \theta^d & \text{if } 1 - \frac{\phi_1}{\theta} - \phi_2 \theta \neq 0 \\ H d \cdot \theta^d & \text{if } 1 - \frac{\phi_1}{\theta} - \phi_2 \theta = 0 \end{cases} \tag{4.21}$$

where A_k and B_k are constants corresponding to the interval $[\tilde{n}_{k-1}, \tilde{n}_k]$, the coefficients ζ_1, ζ_2, D, and H are expressed as follows:

$$\zeta_1 = \frac{1 + \sqrt{1 - 4\phi_1\phi_2}}{2\phi_2}, \zeta_2 = \frac{1 - \sqrt{1 - 4\phi_1\phi_2}}{2\phi_2} \tag{4.22}$$

$$D = \frac{\phi_3}{1 - \phi_1 - \phi_2} \tag{4.23}$$

$$H = \begin{cases} \frac{\phi_4}{1 - \frac{\phi_1}{\theta} - \phi_2 \theta} & \text{if } 1 - \frac{\phi_1}{\theta} - \phi_2 \theta \neq 0 \\ \frac{\phi_4}{\frac{\phi_1}{\theta} - \phi_2 \theta} & \text{if } 1 - \frac{\phi_1}{\theta} - \phi_2 \theta = 0 \end{cases} \tag{4.24}$$

where we define $\phi_1 \triangleq \frac{\gamma q}{1 - \gamma(1-p-q)}$, $\phi_2 \triangleq \frac{\gamma p}{1 - \gamma(1-p-q)}$, $\phi_3 \triangleq \frac{\delta_c}{1 - \gamma(1-p-q)}$, and $\phi_4 \triangleq \frac{\delta_l}{1 - \gamma(1-p-q)}$, and assume that $p \neq 0$ and $q \neq 0$.

Proof. Note that (4.20) is a difference equation [44]. Because we only migrate at states $\{\tilde{n}_k\}$, we have $a(d) = d$ for $d \in (\tilde{n}_{k-1}, \tilde{n}_k)$ $(\forall k)$. From (4.20), for $d \in (\tilde{n}_{k-1}, \tilde{n}_k)$, we have

$$V(d) = \delta_c + \delta_l \theta^d + \gamma \sum_{d_1 = d-1}^{d+1} P_{dd_1} V(d_1) \,. \tag{4.25}$$

We can rewrite (4.25) as

$$V(d) = \phi_1 V(d-1) + \phi_2 V(d+1) + \phi_3 + \phi_4 \theta^d. \tag{4.26}$$

This difference function has characteristic roots as expressed in (4.22). When

$0 < \gamma < 1$, we always have $\zeta_1 \neq \zeta_2$ under the assumption that $p \neq 0$ and $q \neq 0$. Fixing an index k, for $d \in (\tilde{n}_{k-1}, \tilde{n}_k)$, the homogeneous equation of (4.26) has general solution

$$V_h(d) = A_k \zeta_1^d + B_k \zeta_2^d \,. \tag{4.27}$$

To solve the non-homogeneous equation (4.26), we try a particular solution in the form of

$$V_p(d) = \begin{cases} D + H \cdot \theta^d & \text{if } 1 - \frac{\phi_1}{\theta} - \phi_2 \theta \neq 0 \\ D + Hd \cdot \theta^d & \text{if } 1 - \frac{\phi_1}{\theta} - \phi_2 \theta = 0 \end{cases} \tag{4.28}$$

where D and H are constant coefficients. By substituting (4.28) into (4.26), we get (4.23) and (4.24).

Because the expression in (4.26) is related to $d-1$ and $d+1$, the result also holds for the closed interval. Therefore, $V(d)$ can be expressed as (4.21) for $d \in [\tilde{n}_{k-1}, \tilde{n}_k]$ $(\forall k)$. □

The above proposition is subject to the assumption that $p \neq 0$ and $q \neq 0$. When $p = 0$ or $q = 0$, we will have $m_1 = m_2$, in which case we can still find the solution to the difference equation (4.20) using a similar approach, but we omit the discussion for brevity.

We also note that for two different states d_1 and d_2, if policy π has actions $a_\pi(d_1) = d_2$ and $a_\pi(d_2) = d_2$, then

$$V_\pi(d_1) = c_m \left(|d_1 - d_2|\right) + V_\pi(d_2) \,. \tag{4.29}$$

Finding the Coefficients: The coefficients A_k and B_k are unknowns in the solution (4.21) that need to be found using additional constraints. Their values may be different for different k. After A_k and B_k are determined, (4.21) holds for all $d \in [0, N]$.

We assume $1-\frac{\phi_1}{\theta}-\phi_2\theta \neq 0$ and $1-\frac{\phi_2}{\theta}-\phi_1\theta \neq 0$ in the following, the other cases can be derived in a similar way and are omitted for brevity.

Coefficients for interval $[0, \tilde{n}_0]$*:* We have one constraint from the balance equation (4.20) for $d = 0$, which is

$$V(0) = \gamma p_0 V(1) + \gamma(1-p_0)V(0). \tag{4.30}$$

By substituting (4.21) into (4.30), we get

$$A_0(1-\phi_0\zeta_1) + B_0(1-\phi_0\zeta_2) = D(\phi_0 - 1) + H(\phi_0\theta - 1) \tag{4.31}$$

where $\phi_0 \triangleq \frac{\gamma p_0}{1-\gamma(1-p_0)}$. We have another constraint by substituting (4.21) into (4.29), which gives

$$\begin{aligned} &A_0\left(\zeta_1^{\tilde{n}_0} - \zeta_1^{a(\tilde{n}_0)}\right) + B_0\left(\zeta_2^{\tilde{n}_0} - \zeta_2^{a(\tilde{n}_0)}\right) \\ &= \beta_c + \beta_l \mu^{\tilde{n}_0 - a(\tilde{n}_0)} - H\left(\theta^{\tilde{n}_0} - \theta^{a(\tilde{n}_0)}\right). \end{aligned} \tag{4.32}$$

We can find A_0 and B_0 from (4.31) and (4.32).

Coefficients for interval $[\tilde{n}_{k-1}, \tilde{n}_k]$*:* Assume that we have found $V(d)$ for all $d \leq \tilde{n}_{k-1}$. By letting $d = \tilde{n}_{k-1}$ in (4.21), we have the first constraint given by

$$A_k\zeta_1^{\tilde{n}_{k-1}} + B_k\zeta_2^{\tilde{n}_{k-1}} = V(\tilde{n}_{k-1}) - D - H\cdot\theta^{\tilde{n}_{k-1}}. \tag{4.33}$$

For the second constraint, we consider two cases. If $a(\tilde{n}_k) \leq \tilde{n}_{k-1}$, then

$$\begin{aligned} &A_k\zeta_1^{\tilde{n}_k} + B_k\zeta_2^{\tilde{n}_k} \\ &= \beta_c + \beta_l\mu^{\tilde{n}_k - a(\tilde{n}_k)} + V(a(\tilde{n}_k)) - D - H\cdot\theta^{\tilde{n}_k}. \end{aligned} \tag{4.34}$$

If $\tilde{n}_{k-1} < a(\tilde{n}_k) \leq \tilde{n}_k - 1$, then

$$\begin{aligned} & A_k \left(\zeta_1^{\tilde{n}_k} - \zeta_1^{a(\tilde{n}_k)}\right) + B_k \left(\zeta_2^{\tilde{n}_k} - \zeta_2^{a(\tilde{n}_k)}\right) \\ & = \beta_c + \beta_l \mu^{\tilde{n}_k - a(\tilde{n}_k)} - H\left(\theta^{\tilde{n}_k} - \theta^{a(\tilde{n}_k)}\right). \end{aligned} \tag{4.35}$$

The values of A_k and B_k can be solved from (4.33) together with either (4.34) or (4.35).

Solution is in Closed-Form: We note that A_0 and B_0 can be expressed in closed-form, and A_k and B_k for all k can also be expressed in closed-form by substituting (4.21) into (4.33) and (4.34) where necessary. Therefore, (4.21) is a *closed-form solution* for all $d \in [0, N]$. Numerically, we can find $V(d)$ for all $d \in [0, N]$ in $O(N)$ time.

4.4.2.3 Algorithm for Finding the Optimal Policy

Standard approaches to solving for the optimal policy of an MDP include value iteration and policy iteration [41, Chapter 6]. Value iteration finds the optimal policy from Bellman's equation (4.3) iteratively, which may require a large number of iterations before converging to the optimal result. Policy iteration generally requires a smaller number of iterations, because, in each iteration, it finds the exact values of the discounted sum cost $V(d)$ for the policy resulting from the previous iteration, and performs the iteration based on the exact $V(d)$ values. However, in general, the exact $V(d)$ values are found by solving a system of linear equations, which has a time-complexity of $O(N^3)$ when using Gaussian-elimination.

We propose a modified policy-iteration approach for finding the optimal policy, which uses the above result instead of Gaussian-elimination to compute $V(d)$, and also only checks for migrating to lower states or not migrating (according to Proposition 4.1). The algorithm is shown in Algorithm 4.2, where Lines 4–9 find the values

Algorithm 4.2 Modified policy-iteration algorithm based on difference equations

1: Initialize $a(d) = 0$ for all $d = 0, 1, 2, ..., N$
2: Find constants ϕ_0, ϕ_1, ϕ_2, ϕ_3, ϕ_4, ζ_1, ζ_2, D, and H
3: **repeat**
4: $k \leftarrow 0$
5: **for** $d = 1...N$ **do**
6: **if** $a(d) \neq d$ **then**
7: $\tilde{n}_k \leftarrow d$, $k \leftarrow k + 1$
8: **end if**
9: **end for**
10: **for** all $\tilde{n}_k$ **do**
11: **if** $k = 0$ **then**
12: Solve for A_0 and B_0 from (4.31) and (4.32)
13: Find $V(d)$ with $0 \leq d \leq \tilde{n}_k$ from (4.21) with A_0 and B_0 found above
14: **else if** $k > 0$ **then**
15: **if** $a(\tilde{n}_k) \leq \tilde{n}_{k-1}$ **then**
16: Solve for A_k and B_k from (4.33) and (4.34)
17: **else**
18: Solve for A_k and B_k from (4.33) and (4.35)
19: **end if**
20: Find $V(d)$ with $\tilde{n}_{k-1} < d \leq \tilde{n}_k$ from (4.21) with A_k and B_k found above
21: **end if**
22: **end for**
23: **for** $d = 1...N$ **do**
24: $a_{\text{prev}}(d) = a(d)$
25: $a(d) = \arg\min_{a \leq d} \left\{ C_a(d) + \gamma \sum_{j=a-1}^{a+1} P_{aj} V(j) \right\}$
26: **end for**
27: **until** $a_{\text{prev}}(d) = a(d)$ for all d
28: **return** $a(d)$ for all d

of $\tilde{n}_k$, Lines 10–22 find the discounted sum cost values, and Lines 23–26 update the optimal policy. The overall time-complexity for each iteration is $O(N^2)$ in Algorithm 4.2, which reduces time-complexity because standard[5] policy iteration has complexity $O(N^3)$, and the standard value iteration approach does not compute the exact value function in each iteration and generally has long convergence time.

[5]We use the term "standard" here to distinguish from the modified policy iteration mechanism proposed in Algorithm 4.2.

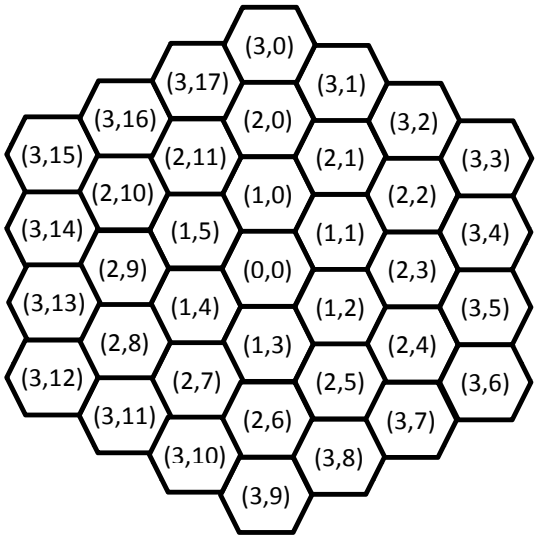


Figure 4.9: Example of 2-D offset model on hexagon cells, where $N = 3$.

4.4.3 Approximate Solution for 2-D Mobility Model

In this section, we show that the distance-based MDP can be used to find a near-optimal service migration policy, where the user conforms to a uniform 2-D random walk mobility model in an infinite space. This mobility model can be used as an approximation to real-world mobility traces (see Section 4.4.4). We consider a hexagonal cell structure, but the approximation procedure can also be used for other 2-D mobility models (such as Manhattan grid) with some parameter changes. The user is assumed to transition to one of its six neighboring cells at the beginning of each timeslot with probability $r \leq \frac{1}{6}$, and stay in the same cell with probability $1 - 6r$.

4.4.3.1 Offset-Based MDP

Similar to 4.3, we define the *offset* of the user from the service as a 2-D vector $e(t) = u(t) - h(t)$ (recall that $u(t)$ and $h(t)$ are also 2-D vectors). Due to the space-homogeneity of the mobility model, it is sufficient to model the state of the MDP by $e(t)$ rather than $s(t)$. The distance metric $\|\varphi_1 - \varphi_2\|$ is defined as the minimum number of hops that are needed to reach cell φ_2 from cell φ_1 on the hexagon model.

We name the states with the same value of $\|e(t)\|$ as a *ring*, and express the states

$\{e(t)\}$ with polar indices (i, j), where the first index i refers to the ring index, and the second index j refers to each of the states within the ring, as shown in Fig. 4.9. For $e(t) = (i, j)$, we have $\|e(t)\| = i$. If $u(t) = h(t)$ (i.e., the actual user and service locations (cells) are the same), then we have $e(t) = (0, 0)$ and $\|e(t)\| = 0$.

As in the distance-based MDP, we assume in the 2-D MDP that we always migrate when $\|e(t)\| \geq N$, where N is a design parameter, and we only consider the state space $\{e(t)\}$ with $\|e(t)\| \leq N$. The system operates in the intermediate state $e'(t) = u(t) - h'(t) = a(e(t))$ after taking action $a(e(t))$. The next state $e(t+1)$ is determined probabilistically according to the transition probability $P_{e'(t),e(t+1)}$. We have $P_{e'(t),e(t+1)} = 1 - 6r$ when $e(t+1) = e'(t)$; $P_{e'(t),e(t+1)} = r$ when $e(t+1)$ is a neighbor of $e'(t)$; and $P_{e'(t),e(t+1)} = 0$ otherwise. Note that we always have $e(t) - e'(t) = h'(t) - h(t)$, so the one-timeslot cost is $C_a(e(t)) = c_m(\|e(t) - e'(t)\|) + c_d(\|e'(t)\|)$.

We note that, even after simplification with the offset model, the 2-D offset-based MDP has a significantly larger number of states compared with the distance-based MDP, because for a distance-based model with N states (excluding state zero), the 2-D offset model has $3N^2 + 3N$ states (excluding state $(0, 0)$). Therefore, we use the distance-based MDP proposed in Section 4.4.2 to approximate the 2-D offset-based MDP, which significantly reduces the computational time as shown in Section 4.4.3.4.

4.4.3.2 Approximation by Distance-based MDP

In the approximation, the parameters of the distance-based MDP are chosen as $p_0 = 6r$, $p = 2.5r$, and $q = 1.5r$. The intuition behind the parameter choice is that, at state $(i'_0, j'_0) = (0, 0)$ in the 2-D MDP, the aggregate probability of transitioning to any state in ring $i_1 = 1$ is $6r$, so we set $p_0 = 6r$; at any other state $(i'_0, j'_0) \neq (0, 0)$, the aggregate probability of transitioning to any state in the higher ring $i_1 = i'_0 + 1$

is either $2r$ or $3r$, and the aggregate probability of transitioning to any state in the lower ring $i_1 = i'_0 - 1$ is either r or $2r$, so we set p and q to the median value of these transition probabilities.

To find the optimal policy for the 2-D MDP, we first find the optimal policy for the distance-based MDP with the parameters defined above. Then, we map the optimal policy from the distance-based MDP to a policy for the 2-D MDP. To explain this mapping, we note that, in the 2-D hexagon offset model, there always exists at least one shortest path from any state (i, j) to an arbitrary state in ring i', the length of this shortest path is $|i - i'|$, and each ring between i and i' is traversed once on the shortest path. For example, one shortest path from state $(3, 2)$ to ring $i' = 1$ is $\{(3, 2), (2, 1), (1, 0)\}$. When the system is in state (i, j) and the optimal action from the distance-based MDP is $a^*(i) = i'$, we perform migration on the shortest path from (i, j) to ring i'. If there exist multiple shortest paths, one path is arbitrarily chosen. For example, if $a(3) = 2$ in the distance-based MDP, then we have either $a(3, 2) = (2, 1)$ or $a(3, 2) = (2, 2)$ in the 2-D MDP. With this mapping, the one-timeslot cost $C_a(d(t))$ for the distance-based MDP and the one-timeslot cost $C_a(e(t))$ for the 2-D MDP are the same, because the migration distances in the distance-based MDP and 2-D MDP are the same (thus same migration cost) and all states in the same ring $i' = \|e'(t)\| = d'(t)$ have the same transmission cost $c_d(\|e'(t)\|) = c_d(d'(t))$.

4.4.3.3 Bound on Approximation Error

Error arises from the approximation because the transition probabilities in the distance-based MDP are not exactly the same as that in the 2-D MDP (there is at most a difference of $0.5r$). In this subsection, we study the difference in the discounted sum costs when using the policy obtained from the distance-based MDP and the true optimal policy for the 2-D MDP. The result is summarized as

Proposition 4.5.

Proposition 4.5. *Let $V_{dist}(e)$ denote the discounted sum cost when using the policy from the distance-based MDP, and let $V^*(e)$ denote the discounted sum cost when using true optimal policy of the 2-D MDP, then we have $V_{dist}(e) - V^*(e) \leq \frac{\gamma r k}{1-\gamma}$ for all e, where $k \triangleq \max_x \{c_m(x+2) - c_m(x)\}$.*

Proof. Details of the proof are presented in Appendix C.3. An outline is given here. The proof is completed in three steps. First, we modify the states of the 2-D MDP in such a way that the aggregate transition probability from any state $(i_0', j_0') \neq (0,0)$ to ring $i_1 = i_0' + 1$ (correspondingly, $i_1 = i_0' - 1$) is $2.5r$ (correspondingly, $1.5r$). We assume that we use a given policy on both the original and modified 2-D MDPs, and show a bound on the difference in the discounted sum costs for these two MDPs. In the second step, we show that the modified 2-D MDP is equivalent to the distance-based MDP. This can be intuitively explained by the reason that the modified 2-D MDP has the same transition probabilities as the distance-based MDP when only considering the ring index i, and also, the one-timeslot cost $C_a(e(t))$ only depends on $\|e(t) - a(e(t))\|$ and $\|a(e(t))\|$, both of which can be determined from the ring indices of $e(t)$ and $a(e(t))$. The third step uses the fact that the optimal policy for the distance-based MDP cannot bring a higher discounted sum cost for the distance-based MDP (and hence the modified 2-D MDP) than any other policy. By utilizing the error bound found in the first step twice, we prove the result. □

The error bound is a constant value when all the related parameters are given. It increases with γ. However, the absolute value of the discounted sum cost also increases with γ, so the relative error can remain low.

4.4.3.4 Numerical Evaluation

The error bound derived in Section 4.4.3.3 is a worst-case upper bound of the error. In this subsection, we evaluate the performance of the proposed approximation method numerically, and focus on the average performance of the approximation.

We consider 2-D random walk mobility with randomly chosen parameter r. The maximum user-service distance is set as $N = 10$. The transmission cost function parameters are selected as $\theta = 0.8$, $\delta_c = 1$, and $\delta_l = -1$. With these parameters, we have $\delta_c + \delta_l = 0$, which means that there is no constant portion in the cost function. For the migration cost, we choose $\mu = 0.8$ and fix $\beta_c + \beta_l = 1$ to represent a constant server processing cost for migration. The parameter $\beta_l \leq 0$ takes different values in the simulations, to represent different sizes of data to be migrated.

Similar to Section 4.3.5, we perform simulations in MATLAB on a computer with Intel Core i7-2600 CPU, 8GB memory, and 64-bit Windows 7. We study the computation time, the number of FLOPs, and the discounted sum cost of the proposed approach that is based on approximating the original 2-D MDP with the distance-based MDP.

For the computation time and number of FLOPs, standard value and policy iteration approaches [41, Chapter 6] are used to solve the original 2-D MDP for comparison. The value iteration procedure terminates when the difference in the discounted sum cost between two iterations is smaller than 0.1 or when a maximum number of iterations has been reached. The maximum number of iterations is set to 5, which is a relatively small number in order to avoid excessive simulation time. We will see later that even with this restriction on maximum number of iterations, value iteration performs worse than standard policy iteration in terms of both computation time and number of FLOPs. The standard policy iteration procedure terminates when the optimal actions from two consecutive iterations are the same, and it always returns the true optimal policy and its corresponding cost values.

The discounted sum cost from the proposed approach is compared with the costs from alternative policies, including the true optimal policy from standard policy iteration on the 2-D model, the never-migrate policy which never migrates except when at states in ring $i \geq N$ (in which case the service is migrated to the current location of the user), the always-migrate policy which always migrates to the current user location when the user and the service are at different locations, and the myopic policy that chooses actions to minimize the one-timeslot cost.

The simulations are run with 50 different random seeds, which generate 50 different values of r, and the overall results are shown in Figs. 4.10–4.12 with different values of the discount factor γ.

Reduction in Time-Complexity: Figs. 4.10–4.12 show that the computation time of the proposed method is only about 0.1% of that of standard value or policy iteration. This time reduction is explained as follows. As discussed in Section 4.4.3.2, for a distance-based MDP with N states (excluding state zero), the 2-D MDP has $N_{\text{2-D}} \triangleq 3N^2 + 3N$ states (excluding state $(0, 0)$). When we ignore the complexity of matrix inversion[6] in the policy iteration procedure, the standard value and policy iteration approaches on the 2-D MDP have a complexity of $O(N_{\text{2-D}}^2)$ in each iteration, because the optimal action needs to be found for each state, which requires enumerating all the states and all possible actions for each state (similar to Lines 23–26 of Algorithm 4.2). In the simulations, $N = 10$, so we have $N_{\text{2-D}} = 330$. Recall that the complexity of Algorithm 4.2 used in the proposed approach is $O(N^2)$, so the ratio of the computational complexities of different approaches can be approximated by $\frac{N_{\text{2-D}}^2}{N^2} \approx 10^3$. Therefore, the standard value and policy iteration approaches consume about 10^3 times more computation time compared to the proposed approach.

When considering the number of FLOPs, the magnitude of reduction of the pro-

[6]We ignore the complexity of matrix inversion here because MATLAB performs matrix operations significantly faster than other operations, as discussed in Section 4.3.5.

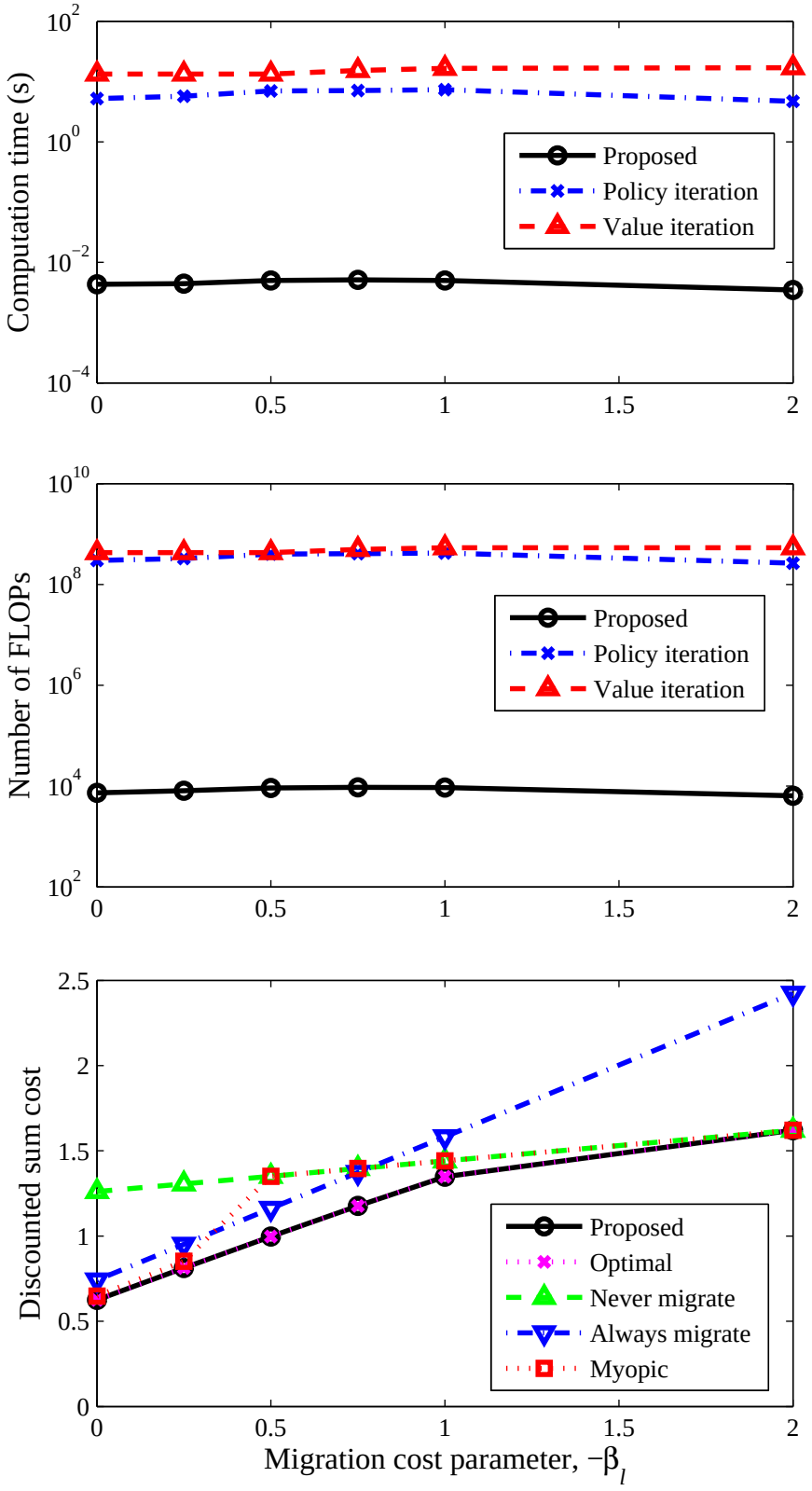


Figure 4.10: Simulation result for 2-D random walk with $\gamma = 0.5$.

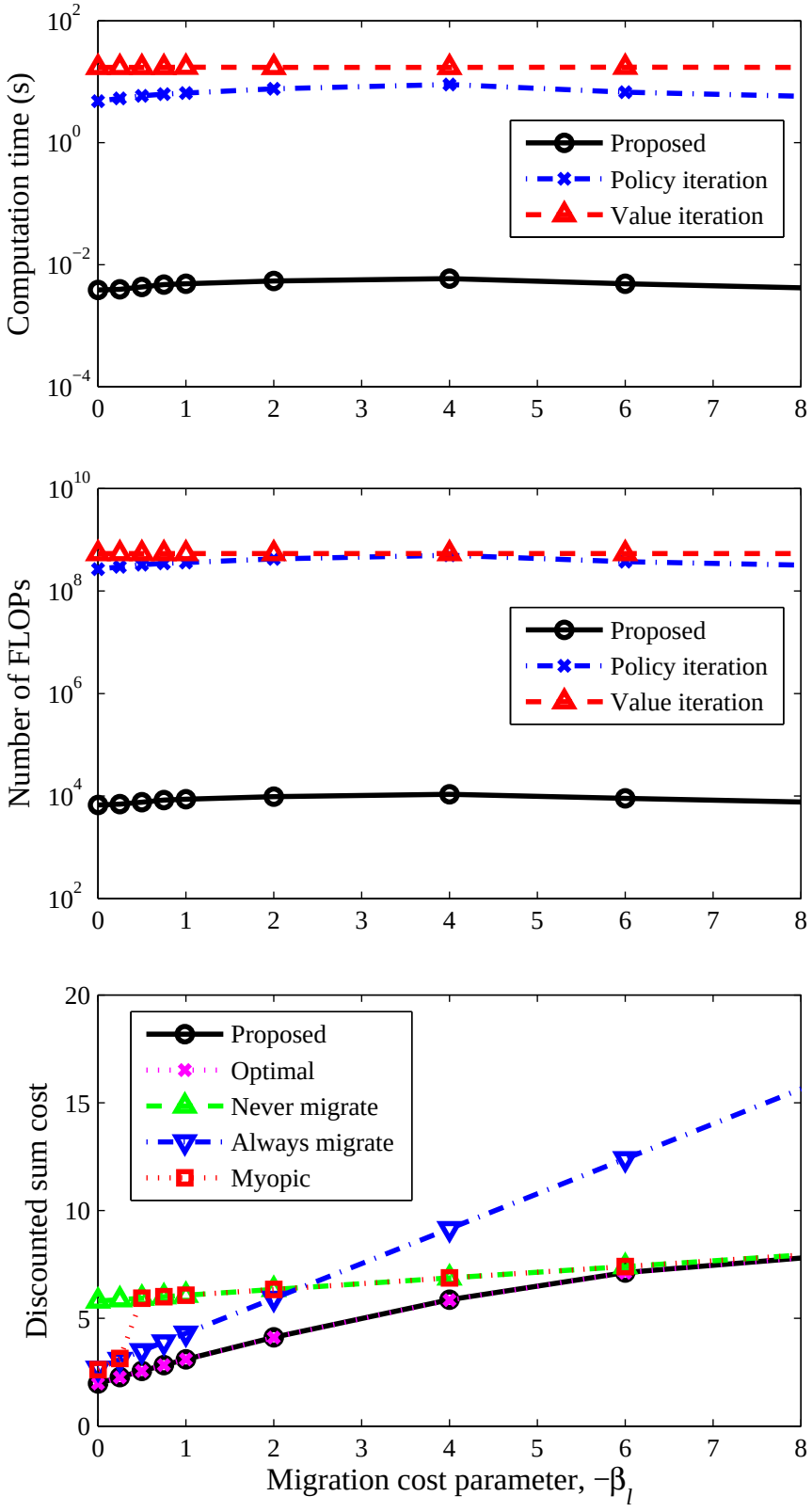


Figure 4.11: Simulation result for 2-D random walk with $\gamma = 0.9$.

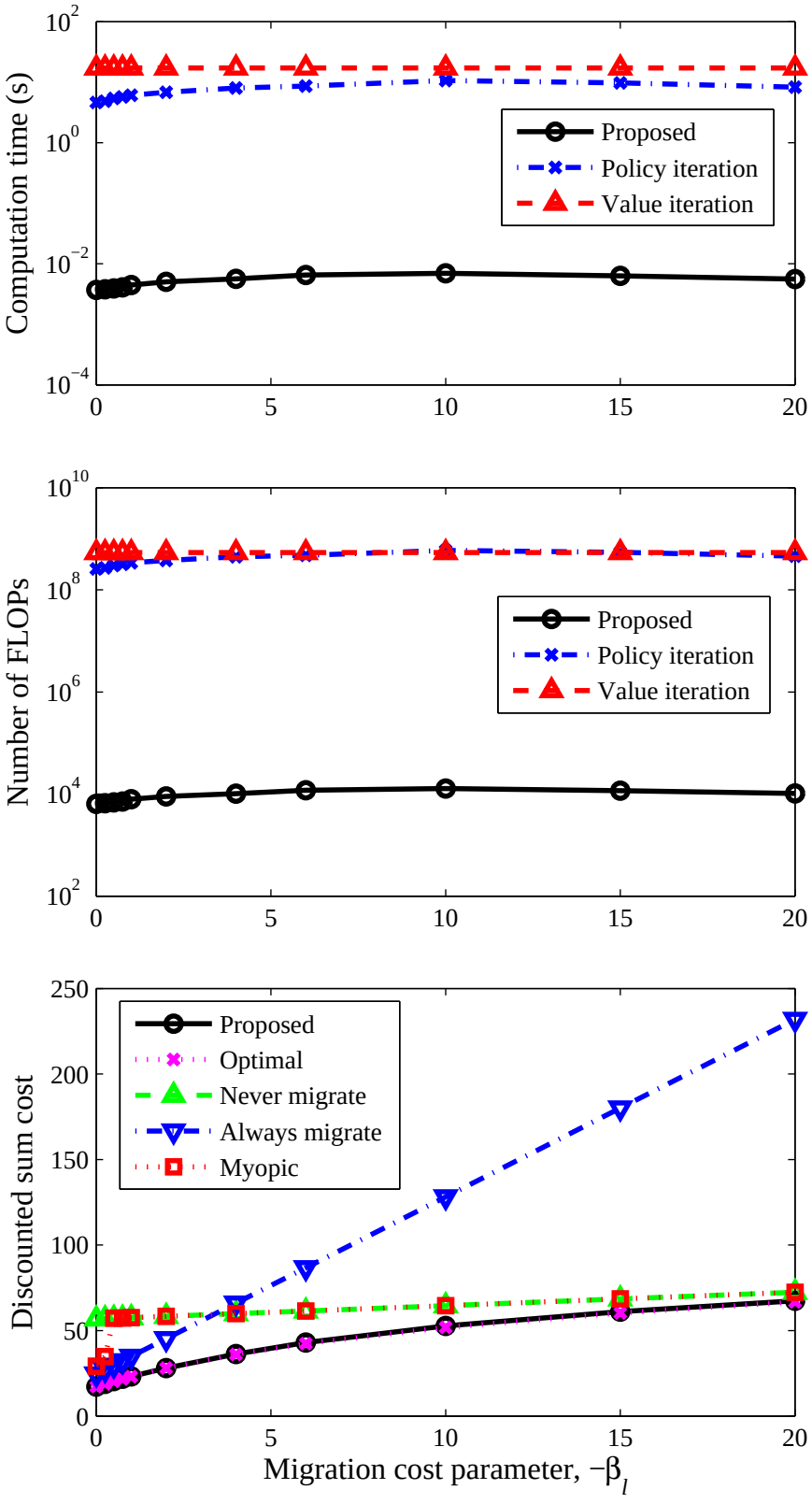


Figure 4.12: Simulation result for 2-D random walk with $\gamma = 0.99$.

posed method compared to standard policy and value iteration approaches ranges between 10^4 and 10^5. This reduction is larger than the reduction in computation time. Reduction in computation time is smaller presumably because of some running time optimization mechanisms applied in MATLAB for performing matrix operations (as discussed in Section 4.3.5) and perhaps also some other operations. Note that the standard policy iteration includes a matrix inversion operation, which has a complexity of $O(N_{\text{2-D}}^3)$ when using Gaussian elimination. This brings an approximate complexity ratio of $\frac{N_{\text{2-D}}^3}{N^2} \approx 10^5$. Different from Section 4.3.5, we no longer see a substantial change in the number of FLOPs under different values of $|\beta_l|$, because our proposed approach here does not include matrix inversion, and the transition probability matrix of the 2-D MDP is less well-structured compared to that of the 1-D MDP (in Section 4.3) so presumably no efficient matrix inversion mechanism exists in this case.

Another aspect different from Section 4.3.5 is that, here in Figs. 4.10–4.12, we do not see a substantial change in the time-complexity of value iteration under different values of γ. The main reason is that we only allow at most 5 iterations in the value iteration approach here as discussed earlier in this section. This is mainly to avoid excessive time we need to run the simulation, because the 2-D scenario is much more complex than the 1-D scenario.

Near-Optimal Cost: We can also see from Figs. 4.10–4.12 that the proposed method yields a discounted sum cost that is very close to the optimal cost. The results also provide several insights into the performance of the baseline policies, which are similar to our observations in the 1-D scenario in Section 4.3.5. Specifically, the cost of the always-migrate policy approximates the optimal cost when $|\beta_l|$ is small, and the cost of the never-migrate policy approximates the optimal cost when $|\beta_l|$ is large. This is because, when $|\beta_l|$ is small, the migration

cost is relatively small, and migration can be beneficial for most cases; when $|\beta_l|$ is large, the migration cost is large, and it is better not to migrate in most cases. The myopic policy is the same as the never-migrate policy when $|\beta_l| \geq 0.5$, because $c_m(x)$ and $c_d(y)$ are both concave according to the simulation settings and we always have $c_m(x) \geq c_d(y)$ when $|\beta_l| \geq 0.5$, where we recall that the myopic policy does not consider the future impact of actions.

There is an intersection of the costs from never-migrate and always-migrate policies. When $|\beta_l|$ takes values that are larger than the value at the intersection point, the optimal cost is close to the cost from the never-migrate policy when γ is small, and the gap becomes larger with a larger γ. The reason is that the benefits of migration become more significant when we look farther ahead into the future. We also note that the cost of never-migrate policy slightly increases as $|\beta_l|$ increases, because the never-migrate policy also occasionally migrates when the user-service distance is greater than or equal to N (see earlier definition).

4.4.4 Application to Real-World Scenarios

In this section, we discuss how the aforementioned approaches can be applied to service migration in the real world, where *multiple users and services co-exist* in the cloud system. We note that in practical scenarios, MMCs may *not* be deployed at every basestation, and each MMC may have a *capacity limit* that restricts the number of services it can host. Theoretically, it is still possible to formulate the service migration problem with these additional constraints as an MDP. However, the resulting MDP will have a significantly larger state space than our current model, and it is far more difficult to solve or approximate this new MDP. While we leave the theoretical analysis of this new MDP as future work, we propose a heuristic approach in this section to handle these additional constraints. The proposed approach is largely based on the results and intuitions obtained in previous sections. The 2-D MDP

approximation approach proposed in Section 4.4.3.2 is used as a subroutine in the heuristic scheme, and the distance-based MDP resulting from the approximation is solved using Algorithm 4.2.

4.4.4.1 Mapping between Real-World and MDP-Model

The mapping between the real-world and the MDP model is discussed as follows.

MMC Controller: We assume that there exists a control entity which we refer to as the *MMC controller*. The MMC controller does not need to be a separate cloud entity. Rather, it can be a service running at one of the MMCs.

Basestations: Each basestation (which may or may not have an MMC connected to it) is assumed to have some basic capability of keeping records on arriving and departing users as well as performing simple monitoring and computational operations.

Timeslots: The physical time length corresponding to one timeslot in the MDP model is a pre-specified parameter, which is a constant for ease of presentation. This parameter can be regarded as a protocol parameter, and it is *not* necessary for all basestations to precisely synchronize on individual timeslots.

Transition Probability: The transition probability parameter r is estimated from the sample paths of multiple users, using the procedure described in Section 4.4.4.2 below. We define a window length $T_w \geq 1$ (represented as the number of timeslots), which specifies the amount of timeslots to look back to estimate the parameter r. We consider the case where r is the same across the whole geographical area, which is a reasonable assumption when different locations within the geographical area under consideration have similarities (for example, they all belong to an urban area). More

sophisticated cases can be studied in the future.

Cost Parameters: The cost parameters β_c, β_l, μ, δ_c, δ_l, and θ are selected based on the actual application scenario, and their values may vary with the background traffic load of the network and MMC servers.

Discount Factor: The discount factor γ can be selected based on the duration of services, and a larger γ corresponds to a longer service duration. Such a selection is because the discount factor γ determines the amount of time to look ahead, if a user only requires the service for a short time, then there is no need to consider the cost for the long-term future. For ease of presentation, we set γ as a constant value. In practice, the value of γ can be different for different users or services.

Policy Update Interval: A policy update interval T_u is defined (represented as the number of timeslots), at which a new migration policy is computed by the MMC controller.

4.4.4.2 Overall Procedure

The data collection, estimation, and service placement procedure is described below.

1. At the beginning of each timeslot, the following is performed:

   (a) Each basestation obtains the identities of its associated users. Based on this information, the basestation computes the number of users that have left the cell (compared to the beginning of the previous timeslot) and the total number of users that are currently in the cell. This information is saved for each timeslot for the duration of T_w and will be used in step 2b.

   (b) The MMC controller collects information on currently active services on each MMC, computes the new placement of services according to the

procedure described in Section 4.4.4.4, and sends the resulting placement instructions to each MMC. The placements of all services are updated based on these instructions.

2. At every interval T_u, the following is performed:

   (a) The MMC controller sends a request to all basestations to collect the current statistics.

   (b) After receiving the request, each basestation n computes the empirical probability of users moving outside of the cell as

$$f_n = \frac{1}{T_w} \sum_{\tau=t-T_w}^{t-1} \frac{m'_n(\tau)}{m_n(\tau)} \tag{4.36}$$

   where the total number of users that are associated to basestation n in slot τ is $m_n(\tau)$, among which $m'_n(\tau)$ users have disconnected from basestation n at the end of slot τ and these users are associated to a different basestation in slot $\tau + 1$; and t denotes the current timeslot index. These empirical probabilities are sent together with other monitored information, such as the current load of the network and MMC server (if the basestation has an MMC connected to it), to the MMC controller.

   (c) After the controller receives responses from all basestations, it performs the following:

      i. Compute the transmission and migration cost parameters β_c, β_l, μ, δ_c, δ_l, and θ based on the measurements at basestations.

      ii. Compute the average of empirical probabilities f_n by

$$\overline{f} = \frac{1}{N_{\text{BS}}} \sum_{n \in \mathcal{N}_{\text{BS}}} f_n \tag{4.37}$$

where $\mathcal{N}_{\text{BS}}$ is the set of basestations and $N_{\text{BS}} = |\mathcal{N}_{\text{BS}}|$ is the total number of basestations. Then, estimate the parameter r by

$$\hat{r} = \overline{f}/6. \tag{4.38}$$

iii. In the distance-based MDP, set $p_0 = 6\hat{r}$, $p = 2.5\hat{r}$, and $q = 1.5\hat{r}$ (as discussed in Section 4.4.3.2), compute and save the optimal distance-based policy from Algorithm 4.2. Also save the estimated cost parameters and the optimal discounted sum costs $V^*(d)$ for all distances d for later use.

Remark: In the procedure presented above, we have assumed that $m_n(\tau) \neq 0$ for all n and τ. This is only for ease of presentation. When there exist some n and τ such that $m_n(\tau) = 0$, we can simply ignore those terms (set the corresponding terms to zero) in the sums of (4.36) and (4.37), and set the values of T_w and N_{BS} to the actual number of terms that are summed up in (4.36) and (4.37), respectively. This simplification does not affect our analysis below if we make a similar substitution in the analysis for cases with $m_n(\tau) = 0$. Thus, we still assume that $m_n(\tau) \neq 0$ for all n and τ in the discussion presented next.

4.4.4.3 Discussion on the Estimation of Parameter r

As introduced in Section 4.4.3, at every timeslot, each user randomly moves to one of its neighboring cells with probability r and stays in the same cell with probability $1 - 6r$. In the real world, the parameter r is unknown a priori and needs to be estimated based on observations of user movement. Equations (4.36)–(4.38) in the above procedure serve for this estimation purpose, and the resulting $\hat{r}$ is an estimator of r. In the following, we analyze some statistical properties of $\hat{r}$ and discuss the

rationale for using such an estimation approach.

We note that the mobility model presented in Section 4.4.3 is for an infinite 2-D space with an infinite number of cells. In reality, the number of cells is finite. We assume in our analysis that each user stays in the same cell with probability $1-6r$ ($r \leq \frac{1}{6}$) and moves out of its current cell with probability $6r$, no matter whether the cell is at the boundary (such as cells in the outer ring $i=3$ in Fig. 4.9) or not. When a cell is at the boundary, its transition probability to each of its neighboring cells is larger than r, because it has less than six neighbors. For example, in Fig. 4.9, a user in cell $(3,0)$ moves to *each* of its neighboring cells (including $(3,17),(2,0),(3,1)$) with probability $2r$, and the total probability of moving out of the cell is still $6r$. We also assume that the mobility patterns of different users are independent of each other.

Proposition 4.6. *Assume that each user follows 2-D random walk (defined above) with parameter r, then $\hat{r}$ is an unbiased estimator of r, i.e., $\mathbb{E}\{\hat{r}\} = r$.*

Proof. We note that in (4.36), $m_n(\tau)$ and $m'_n(\tau)$ are both random variables respectively representing the total number of users associated to basestation (located in cell) n in slot τ and the number of users that have moved out of cell n at the end of slot τ. The values of $m_n(\tau)$ and $m'_n(\tau)$ are random due to the randomness of user mobility.

According to the mobility model, each user moves out of its current cell at the end of a timeslot with probability $6r$, where we recall that $r \leq \frac{1}{6}$ by definition. Hence, under the condition that $m_n(\tau)=k$, $m'_n(\tau)$ follows the binomial distribution with parameters k and $6r$. Thus, the conditional probability

$$\Pr\{m'_n(\tau)=k'|m_n(\tau)=k\} = \left(\begin{array}{c} k \\ k' \end{array}\right)(6r)^{k'}(1-6r)^{k-k'} \tag{4.39}$$

for all $0 \leq k' \leq k$.

From the expression of the mean of binomially distributed random variables, we know that the conditional expectation

$$\mathbb{E}\left\{m'_n(\tau)|m_n(\tau)\right\} = 6r \cdot m_n(\tau). \tag{4.40}$$

Combining (4.36)–(4.38), we have

$$\hat{r} = \frac{1}{6N_{\text{BS}}T_w} \sum_{n\in\mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{m'_n(\tau)}{m_n(\tau)}. \tag{4.41}$$

We then have

$$\begin{aligned}
\mathbb{E}\left\{\hat{r}\right\} &= \mathbb{E}\left\{\frac{1}{6N_{\text{BS}}T_w} \sum_{n\in\mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{m'_n(\tau)}{m_n(\tau)}\right\} \\
&= \frac{1}{6N_{\text{BS}}T_w} \sum_{n\in\mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \mathbb{E}\left\{\frac{\mathbb{E}\left\{m'_n(\tau)|m_n(\tau)\right\}}{m_n(\tau)}\right\} \\
&= \frac{1}{6N_{\text{BS}}T_w} \sum_{n\in\mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \mathbb{E}\left\{\frac{6r \cdot m_n(\tau)}{m_n(\tau)}\right\} \\
&= \frac{r}{N_{\text{BS}}T_w} N_{\text{BS}}T_w \\
&= r
\end{aligned}$$

where the second equality follows from the law of iterated expectations. □

The fact that $\hat{r}$ is an unbiased estimator of r justifies our estimation approach, which intuitively means that the long-term average of the estimated value $\hat{r}$ should not be too far away from the true value of r.

We analyze the variance of the estimator next. Such analysis is not very easy due to the dependency among different random variables. To make the analysis theoretically tractable, we introduce the following assumption.

Assumption 4.1. *We assume that $m'_n(\tau)$ is independent of $m_n(\tilde{\tau})$, $m_{\tilde{n}}(\tau)$, $m_{\tilde{n}}(\tilde{\tau})$, $m'_n(\tilde{\tau})$, $m'_{\tilde{n}}(\tau)$, and $m'_{\tilde{n}}(\tilde{\tau})$ (where $\tilde{n} \neq n$ and $\tilde{\tau} \neq \tau$) when $m_n(\tau)$ is given.*

Essentially, this assumption says that $m'_n(\tau)$ is only dependent on $m_n(\tau)$. In practice when the number of users is large, this assumption can become close to reality, because the locations of different users are independent of each other, and also because of the Markovian property which says that future locations of a user only depends on its present location (and independent of all past locations when the present location is given).

We define $\mathcal{M} \triangleq \{m_n(\tau) : \forall n \in \mathcal{N}_{\text{BS}}, \tau \in [t - T_w, t - 1]\}$ as the set of number of users at all basestations and all timeslots considered in the estimation.

Proposition 4.7. *Assume that Assumption 4.1 is satisfied and each user follows 2-D random walk (defined at the beginning of Section 4.4.4.3) with parameter r. The variance of estimator $\hat{r}$, under the condition that $\mathcal{M}$ is given, is*

$$\text{Var}\{\hat{r}|\mathcal{M}\} = \frac{r(1-6r)}{6N_{BS}^2 T_w^2} \left(\sum_{n \in \mathcal{N}_{BS}} \sum_{\tau=t-T_w}^{t-1} \frac{1}{m_n(\tau)} \right) \tag{4.42}$$

where the conditional variance $\text{Var}\{\hat{r}|\mathcal{M}\}$ *is defined as*

$$\text{Var}\{\hat{r}|\mathcal{M}\} \triangleq \mathbb{E}\left\{\hat{r}^2 \middle| \mathcal{M}\right\} - \left(\mathbb{E}\left\{\hat{r}|\mathcal{M}\right\}\right)^2. \tag{4.43}$$

Proof. Taking the conditional expectation on both sides of (4.41), we have

$$\begin{aligned}
\mathbb{E}\left\{\hat{r}|\mathcal{M}\right\} &= \mathbb{E}\left\{ \frac{1}{6N_{\text{BS}}T_w} \sum_{n \in \mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{m'_n(\tau)}{m_n(\tau)} \middle| \mathcal{M} \right\} \\
&= \frac{1}{6N_{\text{BS}}T_w} \sum_{n \in \mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{\mathbb{E}\left\{m'_n(\tau)|m_n(\tau)\right\}}{m_n(\tau)} \\
&= \frac{1}{6N_{\text{BS}}T_w} \sum_{n \in \mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{6r \cdot m_n(\tau)}{m_n(\tau)}
\end{aligned}$$

$$= \frac{r}{N_{\text{BS}} T_w} N_{\text{BS}} T_w$$
$$= r \tag{4.44}$$

where the second equality is because $m'_n(\tau)$ is independent of $m_n(\tilde{\tau})$, $m_{\tilde{n}}(\tau)$, and $m_{\tilde{n}}(\tilde{\tau})$ (where $\tilde{n} \neq n$ and $\tilde{\tau} \neq \tau$) when $m_n(\tau)$ is given, according to Assumption 4.1. We thus have $(\mathbb{E}\{\hat{r}|\mathcal{M}\})^2 = r^2$.

We focus on evaluating $\mathbb{E}\{\hat{r}^2|\mathcal{M}\}$ in the following. From (4.41), we have

$$\hat{r}^2 = \frac{1}{36 N_{\text{BS}}^2 T_w^2} \left(\sum_{n \in \mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{m'_n(\tau)}{m_n(\tau)} \right)^2$$
$$= \frac{1}{36 N_{\text{BS}}^2 T_w^2} \left(\sum_{n \in \mathcal{N}_{\text{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{(m'_n(\tau))^2}{(m_n(\tau))^2} + \sum_{\substack{n_1, n_2 \in \mathcal{N}_{\text{BS}}; \\ \tau_1, \tau_2 \in [t-T_w, t-1]; \\ n_1 \neq n_2 \text{ and/or } \tau_1 \neq \tau_2}} \frac{m'_{n_1}(\tau_1)}{m_{n_1}(\tau_1)} \cdot \frac{m'_{n_2}(\tau_2)}{m_{n_2}(\tau_2)} \right). \tag{4.45}$$

We now consider the two parts in (4.45). From the proof of Proposition 4.6, we know that $m'_n(\tau)$ follows the binomial distribution when $m_n(\tau)$ is given. Further, when $m_n(\tau)$ is given, $m'_n(\tau)$ is independent of $m_n(\tilde{\tau})$, $m_{\tilde{n}}(\tau)$, and $m_{\tilde{n}}(\tilde{\tau})$ (where $\tilde{n} \neq n$ and $\tilde{\tau} \neq \tau$), according to Assumption 4.1. Thus, we have

$$\mathbb{E}\left\{ \frac{(m'_n(\tau))^2}{(m_n(\tau))^2} \middle| \mathcal{M} \right\} = \frac{\mathbb{E}\left\{ (m'_n(\tau))^2 \middle| m_n(\tau) \right\}}{(m_n(\tau))^2}$$
$$= \frac{m_n(\tau) \cdot 6r \cdot (1-6r) + 36r^2 (m_n(\tau))^2}{(m_n(\tau))^2}$$
$$= \frac{6r \cdot (1-6r)}{m_n(\tau)} + 36r^2 \tag{4.46}$$

where the second equality is a known result for binomially distributed random vari-

ables. We also have

$$\begin{aligned}
&\mathbb{E}\left\{\frac{m'_{n_1}(\tau_1)}{m_{n_1}(\tau_1)}\cdot\frac{m'_{n_2}(\tau_2)}{m_{n_2}(\tau_2)}\middle|\mathcal{M}\right\}\\
&=\mathbb{E}\left\{\frac{m'_{n_1}(\tau_1)}{m_{n_1}(\tau_1)}\middle|m_{n_1}(\tau_1)\right\}\cdot\mathbb{E}\left\{\frac{m'_{n_2}(\tau_2)}{m_{n_2}(\tau_2)}\middle|m_{n_2}(\tau_2)\right\}\\
&=\frac{\mathbb{E}\left\{m'_{n_1}(\tau_1)|m_{n_1}(\tau_1)\right\}}{m_{n_1}(\tau_1)}\cdot\frac{\mathbb{E}\left\{m'_{n_2}(\tau_2)|m_{n_2}(\tau_2)\right\}}{m_{n_2}(\tau_2)}\\
&=36r^2
\end{aligned}\tag{4.47}$$

for $n_1\neq n_2$ and/or $\tau_1\neq\tau_2$, where the first equality follows from the fact that $m'_{n_1}(\tau_1)$ and $m'_{n_2}(\tau_2)$ are independent when $m_{n_1}(\tau_1)$ and $m_{n_2}(\tau_2)$ are given (according to Assumption 4.1), the last equality follows from (4.40).

We now take the conditional expectation on both sides of (4.45), and substitute corresponding terms with (4.46) and (4.47). This yields

$$\begin{aligned}
&\mathbb{E}\left\{\hat{r}^2\middle|\mathcal{M}\right\}\\
&=\frac{1}{36N_{\text{BS}}^2T_w^2}\left(\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1}\mathbb{E}\left\{\frac{(m'_n(\tau))^2}{(m_n(\tau))^2}\middle|\mathcal{M}\right\}\right.\\
&\qquad\left.+\sum_{\substack{n_1,n_2\in\mathcal{N}_{\text{BS}};\\ \tau_1,\tau_2\in[t-T_w,t-1];\\ n_1\neq n_2\text{ and/or }\tau_1\neq\tau_2}}\mathbb{E}\left\{\frac{m'_{n_1}(\tau_1)}{m_{n_1}(\tau_1)}\cdot\frac{m'_{n_2}(\tau_2)}{m_{n_2}(\tau_2)}\middle|\mathcal{M}\right\}\right)\\
&=\frac{1}{36N_{\text{BS}}^2T_w^2}\left(\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1}\left(\frac{6r\cdot(1-6r)}{m_n(\tau)}+36r^2\right)+\sum_{\substack{n_1,n_2\in\mathcal{N}_{\text{BS}};\\ \tau_1,\tau_2\in[t-T_w,t-1];\\ n_1\neq n_2\text{ and/or }\tau_1\neq\tau_2}}36r^2\right)\\
&=\frac{1}{36N_{\text{BS}}^2T_w^2}\left(6r\cdot(1-6r)\cdot\left(\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1}\frac{1}{m_n(\tau)}\right)+N_{\text{BS}}^2T_w^2\cdot36r^2\right)\\
&=\frac{r(1-6r)}{6N_{\text{BS}}^2T_w^2}\left(\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1}\frac{1}{m_n(\tau)}\right)+r^2.
\end{aligned}\tag{4.48}$$

Subtracting $(\mathbb{E}\{\hat{r}|\mathcal{M}\})^2 = r^2$ from the above yields the result. □

Proposition 4.7 gives the conditional variance of estimator $\hat{r}$ when the values of $m_n(\tau)$ are given. It is not straightforward to remove the condition, because it is hard to find the stationary distribution of user locations in a hexagonal 2-D mobility model with a finite number of cells. We note that the set of $m_n(\tau)$ values represents the set of samples in our estimation problem. In standard estimation problems, the sample size is usually deterministic, while it is random in our problem due to random user locations. This causes the difficulty in finding the unconditional variance of our estimator.

However, Proposition 4.7 is important because it gives us a sense on how large the gap between $\hat{r}$ and r is, provided that each user precisely follows the random walk mobility model. We also have the following corollary which gives an upper bound of the unconditional variance.

Corollary 4.3. *Assume that Assumption 4.1 is satisfied and each user follows 2-D random walk (defined at the beginning of Section 4.4.4.3) with parameter r. The unconditional variance of estimator $\hat{r}$ has the following upper bound:*

$$\mathrm{Var}\{\hat{r}\} \leq \frac{1}{144 N_{BS} T_w}. \tag{4.49}$$

Proof. As discussed in Section 4.4.4.2, we assume that $m_n(\tau) \neq 0$ for all n and τ. Thus, we always have $\frac{1}{m_n(\tau)} \leq 1$ (since $m_n(\tau)$ is a positive integer) and

$$\sum_{n \in \mathcal{N}_{\mathrm{BS}}} \sum_{\tau=t-T_w}^{t-1} \frac{1}{m_n(\tau)} \leq N_{\mathrm{BS}} T_w. \tag{4.50}$$

We also have the following bound:

$$r(1-6r) \leq \frac{1}{24} \tag{4.51}$$

for any $r \in [0, \frac{1}{6}]$. The law of total variance gives

$$\mathrm{Var}\{\hat{r}\} = \mathbb{E}\left\{\mathrm{Var}\{\hat{r}|\mathcal{M}\}\right\} + \mathrm{Var}\{\mathbb{E}\left\{\hat{r}|\mathcal{M}\right\}\}$$

According to (4.44), $\mathbb{E}\left\{\hat{r}|\mathcal{M}\right\} = r$ which is a constant, thus $\mathrm{Var}\{\mathbb{E}\left\{\hat{r}|\mathcal{M}\right\}\} = 0$. Therefore, we have

$$\mathrm{Var}\{\hat{r}\} = \mathbb{E}\left\{\mathrm{Var}\{\hat{r}|\mathcal{M}\}\right\} \leq \max_{\mathcal{M}} \mathrm{Var}\{\hat{r}|\mathcal{M}\} \leq \frac{1}{144 N_{\mathrm{BS}} T_w}$$

where the last inequality follows from substituting (4.50) and (4.51) into (4.42). □

We see from Corollary 4.3 that the upper bound of variance is inversely proportional to T_w. This is an intuitive but also very favorable property, which says that when users follow an ideal random walk mobility model with parameter r, we can estimate the value of r as accurate as possible if we have a sufficient amount of samples.

Different from many estimation problems where it is costly (requiring human participation, data communication, etc.) to obtain samples, it is not too difficult to collect samples in our case, because each basestation can save user records at every timeslot, and we can easily adjust the number of samples (proportional to T_w) by changing the amount of timeslots to search back in the record. Therefore, unlike many other estimation problems where the goal is to minimize the variance of estimator under a given sample size, we do not target this goal here. A much more important issue in our problem is that the ideal random walk mobility model may not hold in practice. The parameter estimation procedure in Section 4.4.4.2 conceptually considers such potential model violations, and the rationale of this estimation approach is explained below.

We see in the procedure that we first focus on a particular cell n and compute

the empirical probability of users moving out of that cell in (4.36), by treating each timeslot with equal weight. Then, we take the average of such empirical probabilities of all cells in (4.37). We note that the number of users in different cells and timeslots may be imbalanced. Thus, in (4.36), the empirical probabilities for different cells and slots may be computed with different number of users (samples), i.e., different values of $m_n(\tau)$.

Suppose we think of an alternative approach which first computes the total number of users in all cells and all slots (i.e., $\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1} m_n(\tau)$) and the aggregated total number of users leaving their current cells at the end of slots (i.e., $\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1} m'_n(\tau)$). Then, this approach estimates r by the overall empirical probability of users moving out of their current cells (i.e., $\frac{\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1} m'_n(\tau)}{6\cdot\sum_{n\in\mathcal{N}_{\text{BS}}}\sum_{\tau=t-T_w}^{t-1} m_n(\tau)}$). Intuitively, this alternative estimator may bring a lower variance compared to the current estimator $\hat{r}$, because it treats all the samples as a whole.

However, note that the uniform random walk mobility model *may not hold* precisely in practice; even if it holds, the parameter r *may be time-varying or different in different geographical areas*. Therefore, the proposed approach computes the empirical probability for each cell in each slot first, so that different cells and slots are equally weighted in the estimation, although some cells may have more users than others in certain slots. This is for the consideration of fairness among different cells, so that the performance at different cells remains similar. It is also to avoid statistics at a single slot dominating the overall result. Since r may be time-varying in practice, it is possible that single-slot statistics do not represent the overall case.

4.4.4.4 Service Placement Update

The service placement is updated at the beginning of every timeslot. For each service, if its new location is different from its previous location, a migration of this service is carried out.

The policy found from the MDP model specifies which cell to migrate to when the system is in a particular state $(u(t), h(t))$. However, we may not be able to apply the policy directly, because not every basestation has an MMC connected to it and each MMC has a capacity limit. We may need to make some modifications to the service placement specified by the policy, so that the practical constraints are not violated. The MDP model also does not specify where to place the service if it was not present in the system before. In the following, we present a method to determine the service placement with these practical considerations, which is guided by the optimal policy obtained from the MDP model and at the same time satisfies the practical constraints.

In the proposed procedure, we first ignore the MMC capacity limit and repeat the process presented in steps I and II below for every service. Then, we take into account the capacity limit, and reassign the locations of some services (in step III) that were previously assigned to an MMC whose capacity is exceeded.

(I) **Initial Service Placement:** When the service was not running in the system before (i.e., it is being initialized), the service is placed onto an MMC that has the smallest distance to the user. The intuition for this rule is that the initialization cost and the cost of further operation is usually small with such a placement.

(II) **Dynamic Service Migration:** When the service has been initialized earlier and is currently running in the system, a decision on whether and where to migrate the service is made. Without loss of generality, assume that the current timeslot is $t = 0$. We would like to find an action a that is the solution to the following problem:

$$\min_{a} \; C_a(d(0)) + \gamma \sum_{d(1)=a(d(0))-1}^{a(d(0))+1} P_{a(d(0)),d(1)} V^*(d(1)) \tag{4.52}$$

$$s.t. \text{ there exists an MMC such that the user-service distance is } a(d(0)) \text{ after migration}$$

where $V^*(d)$ stands for the optimal discounted sum cost found from step 2(c)iii in Section 4.4.4.2. We note that (4.52) is a one-step value iteration following the balance equation (4.20). Intuitively, it means that assuming the optimal actions are taken in future slots, find the action for the current slot that incurs the lowest discounted sum cost (including both immediate and future cost). When all basestations have MMCs connected to them, the solution a to problem (4.52) is the same as the optimal action found from step 2(c)iii in Section 4.4.4.2. However, a may be different from the optimal action when some basestations are not connected to MMCs, because there may not exist an MMC that satisfies the distance specification according to action a. The resulting distance-based migration action can be mapped to a migration action on 2-D space using the procedure in Section 4.4.3.2.

(III) **Relocate Service(s) if MMC's Capacity Exceeded:** The placement decisions in steps I and II above do not consider the capacity limit of each MMC, so it is possible that we find the MMC capacity is exceeded after following the steps in the above sections. When this happens, we start with an arbitrary MMC (denoted by i_0) whose capacity constraint is violated. We rank all services in this MMC according to the objective function in (4.52), and start to relocate the service with the highest objective function value. This service is placed on an MMC that still has capacity for hosting it, where the placement decision is also made according to (4.52) but only the subset of MMCs that are still capable of hosting this service are considered. The above process is repeated until the number of services hosted at MMC i_0 is within its capacity limit. Then, this whole process is repeated for other MMCs that have violated capacity

constraints.

Remark: We note that service relocation does not really occur in the system. It is only an intermediate step in the process of finding new service locations. We use this two-step approach involving temporary placement and relocation instead of an alternative one-step approach that checks for MMC capacity when performing the placement/migration in steps I and II, because with such a two-step approach, we can leave the low-cost services within the MMC and move high-cost services to an alternative location.

4.4.4.5 Trace-Driven Simulation

We perform simulation with real-world mobility traces of 536 taxis in San Francisco, collected on May 31, 2008 [45, 46], where different number of taxis are operating (i.e., active) at different time of the day. Each active taxi is modeled as a user that requests a service that is independent of the services of the other taxis. A hexagonal cell structure with 500 m cell separation and $N_{\text{BS}} = 331$ cells (basestations) in total is assumed and each taxi is assumed to be connected to the nearest basestation. We set the physical time length for each timeslot as 60 s. The parameters for data collection and policy update are set as $T_u = 1$ slot, and $T_w = 60$ slots. We choose $N = 10$, $\mu = \theta = 0.8$, and $\gamma = 0.9$. From Corollary 4.3, we know that the standard deviation of estimator $\hat{r}$ (for an ideal random walk mobility model) is upper-bounded by $\sqrt{\frac{1}{144 N_{\text{BS}} T_w}} = 0.00059$, which is reasonably small. Unless otherwise specified, there are 100 basestations that have MMCs connected to them and each MMC can host at most 50 services. The basestations with MMCs are evenly distributed among all basestations. Note that the locations of taxis in the dataset are unevenly distributed and the density of taxis can be very high in some areas at certain times of the day.

Similar to Section 4.4.3.4, we compare the performance of the proposed method with baseline policies including always/never-migrate and myopic. To cope with

the nature of the policies, the objective function in (4.52) is modified accordingly for these baseline policies. The objective in (4.52) is defined as the user-service distance for the always- or never-migrate policies (recall that migration also happens with never-migrate policy when the user-service distance is N or larger), and it is defined as the one-timeslot cost for the myopic policy.

Cost definition: It is assumed that the system load is proportional to the number of taxis in operation, and we define parameters R_t and R_p as weighting factors respectively for transmission bandwidth and MMC processing resources. Then, we define $G_t \triangleq 1 \Big/ \left(1 - \frac{m_{\text{cur}}}{R_t m_{\max}}\right)$ and $G_p \triangleq 1 \Big/ \left(1 - \frac{m_{\text{cur}}}{R_p m_{\max}}\right)$, where m_{cur} denotes the number of taxis in operation at the time when the optimal policy is being computed, and $m_{\max}$ denotes the maximum number of taxis that may simultaneously operate at any time instant in the considered dataset. The expressions for G_t and G_p have the same form as the average queuing delay expression in queuing theory [47], and they mimic the delay of data transmission (G_t) and processing (G_p).

The cost parameters are then defined as $\beta_c = G_p + G_t$, $\beta_l = -G_t$, $\delta_c = G_t$, and $\delta_l = -G_t$. With such a definition, we have $\beta_c + \beta_l = G_p$ and $\delta_c + \delta_l = 0$, which means that the constant part of the migration cost is G_p (to represent the processing cost for migration) and there is no constant part in the cost for data transmission. We set $\beta_l = \delta_l = G_t$ because this part of cost can be regarded as related to data transmission in both $c_m(x)$ and $c_d(y)$.

Results: The instantaneous cost in each time slot (i.e., the $C_a(s(t))$ values) averaged over all users that are active in that slot is collected and shown in Fig. 4.13. The fluctuation in cost values is due to different system load over the day, and we can see that the proposed method almost always outperforms the baseline approaches.

To consider the overall performance under different parameter settings, we denote the cost of the proposed method as C and the cost of the baseline method under

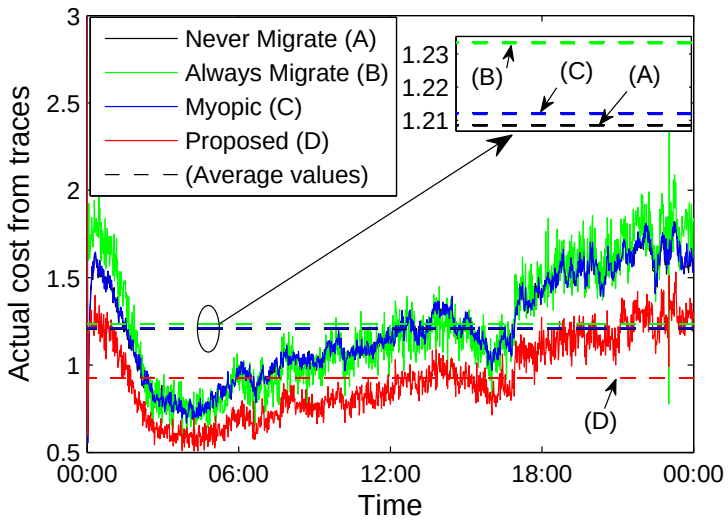


Figure 4.13: Instantaneous average cost per user in each timeslot over a day in trace-driven simulation, where $R_t = R_p = 1.5$. An enlarged plot of the circled area is shown on the top-right of the plot. The arrows annotated with (A), (B), (C), and (D) point to the average values over the whole day of the corresponding policy.

comparison as C_0, and we define the *cost reduction* as $(C_0 - C)/C_0$. Fig. 4.14 shows the cost reductions (averaged over the entire day) under different parameter settings.

From Figs. 4.14(a)–(d), we can see that under the default number of MMCs and capacity limit at each MMC, the proposed approach is beneficial with cost reductions ranging from 9% to 44% compared to the never/always-migrate or myopic policies.

From Figs. 4.14(e)–(f), we see that the cost reductions compared to never-migrate and myopic policies become small in the case where either the number of MMCs is small or the capacity of each MMC is low. In this case, it is hardly possible to migrate the service to a better location because the action space for migration is very small. Therefore, the proposed approach gives a similar cost as baseline policies, but still outperforms them as indicated by a positive cost reduction on average. The cost reduction compared to the always-migrate policy becomes slightly higher in the case of small action space, because the always-migrate policy always incurs migration cost even though the benefit of such migration is not obvious.

The fact that costs fluctuate over the day and vary with different amount of avail-

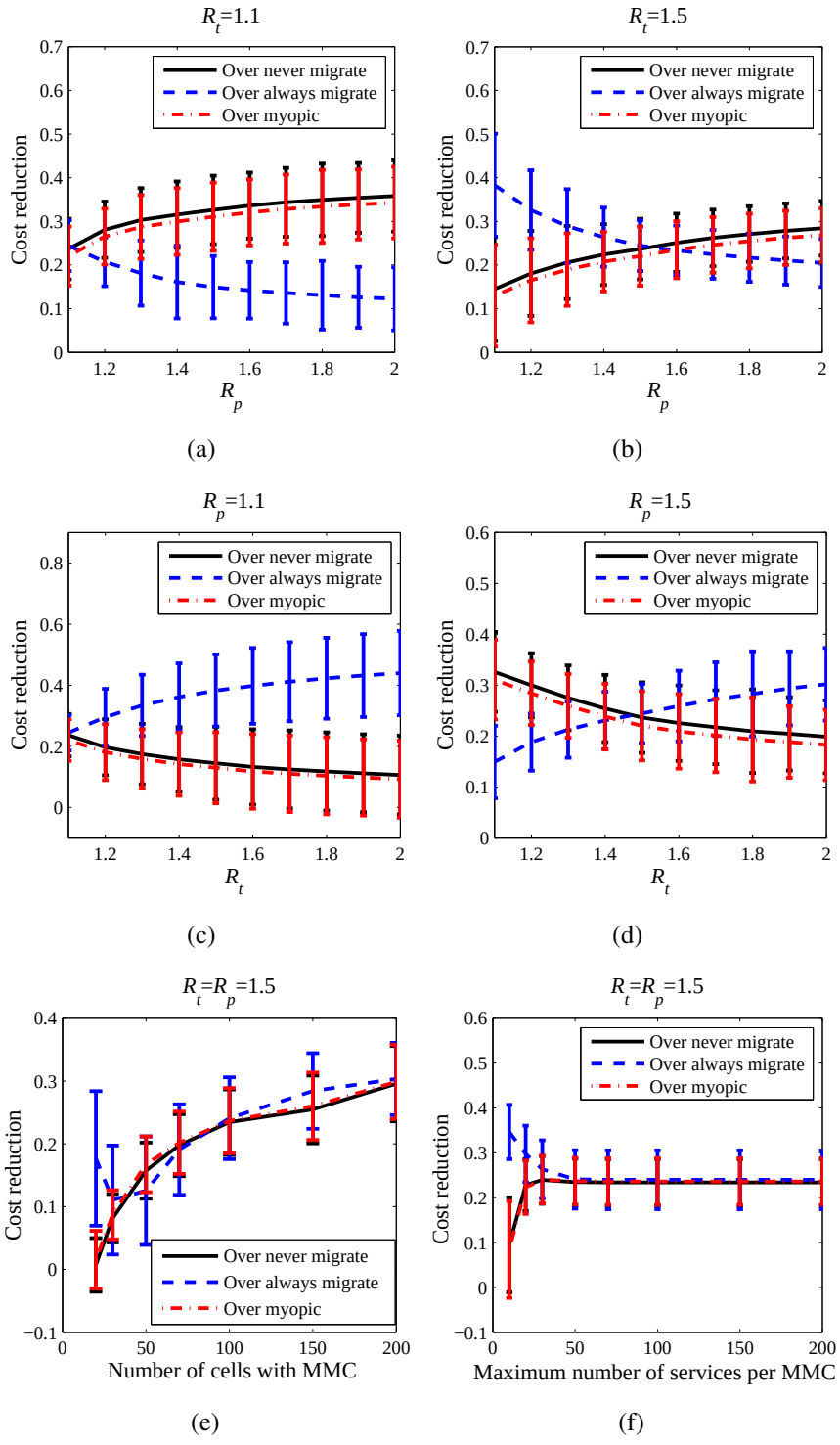


Figure 4.14: Cost reduction (averaged over the entire day) compared to alternative policies in trace-driven simulation, the error bars denote the standard deviation (where we regard the instantaneous cost at different time of the day as samples): (a)–(b) cost reduction vs. different R_p, (c)–(d) cost reduction vs. different R_t, (e) cost reduction vs. different number of cells with MMC, (f) cost reduction vs. different capacity limit of each MMC (expressed as the maximum number of services allowed per MMC).

able resources (which is adjusted by R_t and R_p) also implies that it is necessary to compute the optimal policy in real-time, based on recent observations of the system condition.

4.5 Discussion

Some assumptions have been made in this chapter to make the problem theoretically tractable and also for ease of presentation. In this section, we justify these assumptions from a practical point of view and discuss possible extensions.

Cost Functions: To ease our discussion, we have limited our attention to transmission and migration costs in this chapter. This can be extended to include more sophisticated cost models. For example, the transmission cost can be extended to include the computational cost of hosting the service at an MMC, by adding a constant value to the transmission cost expression. As in Section 4.4.4.5, the cost values can also be time-varying and related to the background system load. Furthermore, the cost definition can be practically regarded as the average cost over multiple locations, which means that when seen from a single location, the monotonicity of cost values with distances does not need to apply. This makes the proposed approach less restrictive in terms of practical applicability.

We also note that it is generally possible to formulate an MDP with additional dimensions in cost modeling, such as one that includes the state of the network, load at each specific MMC, state of the service to avoid service interruption when in critical state, etc. However, this requires a significantly larger state space compared to our formulation in this chapter, as we need to include those network/MMC/service states in the state space of the MDP. There is a tradeoff between the complexity of solving the problem and accuracy of cost modeling. Such issues can be studied in the future, where we envision similar approximation techniques as in this chapter

can be used to approximately solve the resulting MDP.

Single/Multiple Users: Although we have focused on a single user in our problem modeling, practical cases involving multiple users running independent services can be considered by setting cost functions related to the background traffic generated by other users, then considering each user independently, as in the cost definition of the simulation in Section 4.4.4.5. More general cases where each MMC has a capacity limit and not all basestations have MMCs attached to it can be tackled using the heuristic approach presented in Section 4.4.4. From a theoretical perspective, one can also formulate this and other generalized problems as an MDP with larger state space (similar to the generalized cost model discussed above), but we leave it as future work.

Random Walk: The random walk mobility model is used as a modeling assumption, which not only simplifies the theoretical analysis, but also makes the practical implementation of the proposed method fairly simple in the sense that only the empirical probability of users moving outside of the cell needs to be recorded (see Section 4.4.4.2). This model can capture the average mobility of a large number of users. The simulation results in Section 4.4.4.5 confirm that this model provides good performance, even though individual users do not necessarily follow a random walk.

Centralized/Distributed Control: We have focused on a centralized control mechanism in this chapter for ease of presentation. However, many parts of the proposed approach can be performed in a distributed manner. For example, in step 1b in Section 4.4.4.2, the service placement decision can be made among a smaller group of MMCs if the controller sends the results from step 2(c)iii to these MMCs. In particular, the temporary service placement decision in Sections I and II can be

made locally on each MMC (provided that it knows the locations of other MMCs). The capacity violation check in Section III can also be performed locally. If some MMCs are in excess of capacity, service relocation can be performed using a few control messages exchanged between MMCs, where the number of necessary messages is proportional to the number of services to relocate. Relocation would rarely occur if the system load is not very high. The computation of average empirical probability in step 2(c)ii in Section 4.4.4.2 can also be distributed in the sense that a subset of MMCs compute local averages, which are subsequently sent to the MMC controller that computes the global average.

MMC-Basestation Co-location: For ease of presentation, we have assumed that MMCs are co-located with basestations. However, our proposed approach is not restricted to such cases and can easily incorporate scenarios where MMCs are not co-located with basestations as long as the costs are geographically dependent.

4.6 Summary

We have taken an MDP-based approach to dynamic service migration in this chapter. Noting that the state space of the MDP for the original problem can be arbitrarily large, we have made simplifications that are either provably optimal or suboptimal with a provable optimality gap. We have also discussed ways to implement the proposed algorithms in practical systems, and evaluated their performance with both synthetic mobility traces and real-world mobility traces of San Francisco taxis.

CHAPTER 5

Dynamic Service Placement with Predicted Future Costs

5.1 Introduction

The previous chapter presented an MDP-based approach to dynamic service migration. Although the proposed approach is simple, it is constrained to cases where the user mobility follows or can be approximated by a mobility model that can be described by a Markov chain. We note that there are also practical cases where the Markovian assumption is not valid [48]. Furthermore, the definition of costs related to the locations of users and service instances may be inapplicable if the load on different MMCs are imbalanced or if we consider the centralized cloud as a placement option.

In this chapter, we assume that there is an underlying mechanism to predict the future costs to some known accuracy. Using these predicted costs, we consider a more general setting that may have different forms of cost functions, as well as heterogeneous network structure and mobility models. Meanwhile, different from Chapter 4 where we mainly consider services that are constantly running for each user, we consider service instances that may arrive and depart over time.

5.1.1 Related Work

Systems with online (and usually unpredictable) arrivals and departures have been studied in the field of online approximation algorithms [24, 49]. The goal is to design efficient algorithms (usually with polynomial time-complexity) that have reasonable competitive ratios. However, most existing work focuses on problems that can be formulated as integer *linear* programs. Problems that have *convex but non-linear* objective functions have attracted attention only very recently [50, 51], where the focus is on online covering problems in which new constraints arrive over time. Our problem is different from the existing work in the sense that the online arrivals in our problem are abstracted as *change* in constraints (or, with a slightly different but equivalent formulation, adding new *variables*) instead of adding new constraints, and we consider the average cost over multiple timeslots. Meanwhile, online departures are not considered in [50, 51]. We also note that existing online algorithms with provable performance guarantees are often of theoretical nature [24, 49, 50, 51], which may not be straightforward to apply in practical systems. At the same time, most online algorithms applied in practice are of heuristic nature without provable performance guarantees, which may perform poorly particularly under critical settings. We propose a simple and practically applicable online algorithm with provable performance guarantees in this chapter, and also verify its performance with simulation using both synthetic arrivals and real-world user traces.

5.1.2 Main Contributions

In this chapter, we consider a general setting which allows heterogeneity in cost values, network structure, and mobility models. We assume that the cost is related to a finite set of parameters, which can include the locations and preferences of users, load in the system, database locations, etc. We focus on the case where there is an

underlying mechanism to predict the future values of these parameters[1], and also assume that the prediction mechanism provides the most likely future values and an upper bound on possible deviation of the actual value from the predicted value. Such an assumption is valid for many prediction methods that provide guarantees on prediction accuracy. Based on the predicted parameters, the (predicted) future costs of each configuration can be found, in which each configuration represents one particular placement sequence of service instances. While we do not deal with specific cost prediction methods in this thesis, a brief summary of possible approaches we can use is given in Section 5.2.2.

With the above assumption, we formulate a problem of finding the optimal placement sequence of service instances that minimizes the average cost over time. We define a look-ahead window to represent the amount of time that we look (predict) into the future. The main contributions of this chapter are summarized as follows:

1. We first focus on the *offline problem* of service instance placement using the predicted costs within a specific look-ahead window, where we assume that the service instance arrivals and departures within this look-ahead window are known beforehand. We show that this problem is equivalent to a shortest-path problem and propose an algorithm (Algorithm 5.2 in Section 5.3.3) to find its optimal solution, by leveraging dynamic programming techniques.

2. We note that it is often practically infeasible to know in advance about when a service instance will arrive or depart. Meanwhile, Algorithm 5.2 may have exponential time-complexity when there exist multiple service instances, which means that it can be very time consuming to find the solution. Therefore, we propose an *online approximation algorithm* that finds the placement of each

[1]We regard these cost parameters as predictable because they are generally related to the overall state of the system or historical pattern of users, which are unlikely to vary significantly from its previous state or pattern within a short time. This is different from arrivals and departures of service instances, which can be spontaneous and unlikely to follow a predictable pattern.

service instance at the time it arrives. This online algorithm has polynomial time-complexity and can find the solution within a reasonably short time. It calls Algorithm 5.2 as a subroutine on each service instance arrival. We analytically evaluate the performance of this online algorithm compared to the optimal offline placement. The proposed online algorithm is $O(1)$-competitive for certain types of cost functions (including those which are linear, polynomial, or in some other specific form), under some mild assumptions.

3. Considering the existence of prediction errors, we propose a method to find the *optimal look-ahead window size*, such that an upper bound on the actual placement cost is minimized.

4. The effectiveness of the proposed approach is evaluated by *simulations* with both synthetic service instance arrivals and *real-world mobility traces* of San Francisco taxis.

5.2 Problem Formulation

We consider a cloud computing system as shown in Fig. 1.1, where the clouds are indexed by $n \in \{1, 2, ..., N\}$. Each cloud n can be either an MMC or a centralized cloud. All MMCs together with the centralized cloud can host service instances that may arrive and leave the system over time. A service instance is a process that is executed for a particular task of a particular cloud service, which may or may not be embedded into a containing environment (such as virtual machine or Linux container). Each service instance *may serve one or multiple users*, where there usually exists data transfer between the instance and the users it is serving. A time-slotted system as shown in Fig. 5.1 is considered, in which the actual physical time interval corresponding to each slot $t = 1, 2, 3, ...$ can be either the same or different.

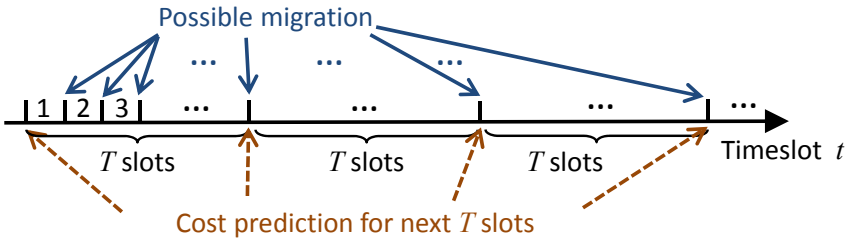


Figure 5.1: Timing of the proposed approach.

We consider a window-based control framework, where every T slots, a controller performs cost prediction and computes the service instance placement sequence for the next T slots. We define these T consecutive slots as a *look-ahead window*. Service instance placement within each window are found either at the beginning of the window (in the offline case) or whenever an instance arrives (in the online case). We limit ourselves within one look-ahead window when finding the placement sequence. In other words, we do not attempt to find the placement in the next window until the time for the current window has elapsed and the next window starts. Our solution can also be extended to a slot-based control framework where the controller computes the next T-slot service placement sequence at the beginning of every slot, based on predicted cost parameters for the next T slots. We leave the detailed comparison of these frameworks and their variations for future work.

5.2.1 Definitions

We introduce some definitions in the following.

5.2.1.1 Service Instances

Service instances may arrive and depart over time. We keep an index counter to assign an index for each new instance. The counter is initialized to zero when the cloud system starts to operate[2]. Upon a service instance arrival, we increment the

[2]This is only for ease of presentation. In practice, the index can be reset when the maximum possible number of the counter is reached.

counter by one, so that if the previously arrived instance has index i, a newly arrived instance will have index $i+1$. With this definition, if $i < i'$, instance i arrives no later than instance i'. A particular instance i can only arrive once, and we assume that arrivals always occur at the beginning of a slot and departures always occur at the end of a slot. For example, consider timeslots $t = 1, 2, 3, 4, 5$, instance $i = 2$ may arrive at the beginning of slot $t = 2$, and depart at the end of slot $t = 4$. At any timeslot t, instance i can have one of the following states: not arrived, running, or departed. For the above example, instance $i = 2$ has not yet arrived in slot $t = 1$, it is running in slots $t = 2, 3, 4$, and it has already departed in slot $t = 5$. Note that an instance can be running across multiple windows each containing T slots before it departs.

5.2.1.2 Configurations

Consider an arbitrary sequence of consecutive timeslots $t \in \{t_0, t_0+1, ..., t_0+Q-1\}$, where Q is an integer. Assume that the instance with the smallest index running in slot t_0 has index i_0, and the instance with the largest index running in *any of* the slots within $\{t_0, ..., t_0 + Q - 1\}$ has index $i_0 + M - 1$. According to the index assignment discussed in Section 5.2.1.1, there can be at most M instances running in any slot $t \in \{t_0, ..., t_0 + Q - 1\}$.

We define a Q-by-M matrix denoted by $\boldsymbol{\pi}$, where its (q, i)th ($q \in \{1, ..., Q\}$) element $(\boldsymbol{\pi})_{qi} \in \{0, 1, 2, ..., N\}$ denotes the location of service instance i in slot $t_q \triangleq t_0 + q - 1$, in which "$\triangleq$" stands for "is defined to be equal to". We set $(\boldsymbol{\pi})_{qi}$ according to the state of instance i in slot t_q, as follows

$$(\boldsymbol{\pi})_{qi} = \begin{cases} 0, & \text{if } i \text{ is not running in slot } t_q \\ n, & \text{if } i \text{ is running in cloud } n \text{ in slot } t_q \end{cases}$$

where instance i is not running if it has not yet arrived or has already departed. The matrix $\boldsymbol{\pi}$ is called the *configuration* of instances in slots $\{t_0, ..., t_0+Q-1\}$. Throughout this chapter, we use matrix $\boldsymbol{\pi}$ to represent configurations in different subsets of timeslots. We will write $\boldsymbol{\pi}(t_0, t_1, ..., t_m)$ to explicitly denote the configuration in slots $\{t_0, t_1, ..., t_m\}$ (and we have $Q = t_m - t_0 + 1$), and we write $\boldsymbol{\pi}$ for short where the considered slots can be inferred from the context. When considering a single slot t, $\boldsymbol{\pi}(t)$ becomes a vector (i.e., $Q = 1$).

Remark: Note that the configurations in different slots can appear either in the same matrix or in different matrices. This means, from $\boldsymbol{\pi}(t_0, ..., t_0 + Q - 1)$, we can get $\boldsymbol{\pi}(t)$ for any $t \in \{t_0, ..., t_0 + Q - 1\}$, as well as $\boldsymbol{\pi}(t-1, t)$ for any $t \in \{t_0 + 1, ..., t_0 + Q - 1\}$, etc., and vice versa. For ease of presentation later in this chapter, we define $(\boldsymbol{\pi}(0))_i = 0$ for any i.

5.2.1.3 Costs

Similar to Chapter 4, we consider two types of costs. The *local cost* $U(t, \boldsymbol{\pi}(t))$ specifies the cost of data transmission (e.g., between each pair of user and service instance) and processing in slot t when the configuration in slot t is $\boldsymbol{\pi}(t)$. Its value can depend on many factors, including user location, network condition, load of clouds, etc., as discussed in Section 5.1.2. When a service instance is initiated in slot t, the local cost in slot t also includes the cost of initial placement of the corresponding service instance(s). We then define the *migration cost* $W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t))$, which specifies the cost related to migration between slots $t-1$ and t, which respectively have configurations $\boldsymbol{\pi}(t-1)$ and $\boldsymbol{\pi}(t)$. There is no migration cost in the very first timeslot (start of the system), thus we define $W(1, \cdot, \cdot) = 0$. The sum of local and migration costs in slot t when following configuration $\boldsymbol{\pi}(t-1, t)$ is given by

$$C_{\boldsymbol{\pi}(t-1,t)}(t) \triangleq U(t, \boldsymbol{\pi}(t)) + W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t)). \tag{5.1}$$

The above defined costs are *aggregated costs* for all clouds and transmission links in the system.

5.2.2 Actual and Predicted Costs

To distinguish between the actual and predicted cost values, for a given configuration $\boldsymbol{\pi}(t-1,t)$, we let $A_{\boldsymbol{\pi}}(t)$ denote the *actual* value of $C_{\boldsymbol{\pi}}(t)$, and let $D_{\boldsymbol{\pi}}^{t_0}(t)$ denote the *predicted* most likely value of $C_{\boldsymbol{\pi}}(t)$, when cost-parameter prediction is performed at the beginning of slot t_0. For completeness of notations, we define $D_{\boldsymbol{\pi}}^{t_0}(t) = A_{\boldsymbol{\pi}}(t)$ for $t < t_0$, because at the beginning of t_0, the costs of all past timeslots are known. For $t \geq t_0$, we assume that the absolute difference between $A_{\boldsymbol{\pi}}(t)$ and $D_{\boldsymbol{\pi}}^{t_0}(t)$ is at most

$$\varepsilon(\tau) \triangleq \max_{\boldsymbol{\pi}(t-1,t),t_0} \left| A_{\boldsymbol{\pi}(t-1,t)}(t_0+\tau) - D_{\boldsymbol{\pi}(t-1,t)}^{t_0}(t_0+\tau) \right|$$

which represents the maximum error when looking ahead for τ slots, among all possible configurations $\boldsymbol{\pi}$ and all possible prediction time instant t_0. The function $\varepsilon(\tau)$ is assumed to be non-decreasing with τ, because we generally cannot have lower error when we look farther ahead into the future. The specific value of $\varepsilon(\tau)$ is assumed to be provided by the cost prediction module.

How to Obtain and Predict Cost Parameters: We note that specific methods for obtaining current cost parameters and predicting their future values are beyond the scope of this thesis, but we anticipate that existing approaches for cloud monitoring and future cost prediction can be applied. We provide a brief discussion on possible approaches one can use.

We start with monitoring the current cloud performance and user locations. The simplest way for cloud monitoring is to send the current parameters that can be directly measured on the system (such as CPU and memory utilization) to the cloud controller. However, such simple methods may either generate high overhead for

control information exchange or be inaccurate due to the possibly abrupt change in computational resource occupation of cloud applications. More systematic approaches for cloud monitoring are discussed in the survey in [52], where it is shown that different monitoring mechanisms target slightly different goals, for example some focus on the accuracy of monitored parameters while others focus on the adaptability to a change in resource occupation. The user locations can be monitored by determining which basestation each user is connected to. This granularity of location is sufficient for the case where MMCs are connected directly to basestations. A simple monitoring approach can notify the cloud controller of user handoff instances between different basestations, so that the controller always keeps track of which basestation each user is associated to.

After the current state of the system has been monitored, the future states (parameters in the cost model) need to be predicted to determine the predicted future cost. One simple approach for this purpose is to measure cost parameters based on current observation, and regard them as parameters for the future cost until the next measurement is taken. The prediction accuracy in this case is related to how fast the cost parameters vary, which can be estimated from historical records of these parameters. There are more intelligent ways for making predictions, which are expected to be more accurate than the simple approach which takes current measurements as future values. For example, for predicting future cloud performance, one can use linear models such as Kalman filters, discrete models such as hidden Markov chains, or ideas from expert systems [53]. For predicting future user locations, one can make use of social network information of users to derive each user's intent of moving to a particular area, and some periodicity of user mobility that can be derived from the user's historical mobility traces [54], for instance.

5.2.3 Our Goal

Our ultimate goal is to find an optimal configuration $\boldsymbol{\pi}^*(1, ..., \infty)$ that minimizes the *actual* average cost over sufficiently long time, i.e.

$$\boldsymbol{\pi}^*(1, ..., \infty) = \arg \min_{\boldsymbol{\pi}(1,...,\infty)} \lim_{T_{\max} \to \infty} \frac{\sum_{t=1}^{T_{\max}} A_{\boldsymbol{\pi}(t-1,t)}(t)}{T_{\max}}. \tag{5.2}$$

However, it is impractical to find the optimal solution to (5.2), because we cannot precisely predict the future costs and also do not have exact knowledge on instance arrival and departure events in the future. Therefore, we focus on obtaining an approximate solution to (5.2) by utilizing *predicted* cost values that are collected every T slots.

Now, the service placement problem includes two separable parts: one is finding the look-ahead window size T, which will be discussed in Section 5.5; the other is finding the placement sequence (configuration) within each window, where we consider both offline and online placements and will be discussed in Section 5.3 (for offline placement) and Section 5.4 (for online placement). The offline placement assumes that at the beginning of window T, we know the exact time of when each instance arrives and departs within the rest of window T. The online placement does not assume this knowledge. We note that the notion of "offline" here does *not* imply exact knowledge of future costs. The predicted costs are used in both offline and online placements.

5.3 Offline Service Placement with Given Look-Ahead Window Size

In this section, we focus on the offline placement problem. We denote the configuration found for this problem by $\boldsymbol{\pi}_{\text{off}}$.

Algorithm 5.1 Procedure for offline service placement

1: Initialize $t_0 = 1$
2: **loop**
3: At the beginning of slot t_0, find the solution to (5.3)
4: Apply placements $\boldsymbol{\pi}_{\text{off}}(t_0, ..., t_0 + T - 1)$ in timeslots $t_0, ..., t_0 + T - 1$
5: $t_0 \leftarrow t_0 + T$
6: **end loop**

5.3.1 Procedure

We start by illustrating the high-level procedure of finding $\boldsymbol{\pi}_{\text{off}}$. When the look-ahead window size T is given, the placement sequence $\boldsymbol{\pi}_{\text{off}}$ is found sequentially for each window (containing timeslots $t_0, ..., t_0 + T - 1$), by solving the following optimization problem:

$$\boldsymbol{\pi}_{\text{off}}(t_0, ..., t_0+T-1) = \arg \min_{\boldsymbol{\pi}(t_0,...,t_0+T-1)} \sum_{t=t_0}^{t_0+T-1} D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t) \tag{5.3}$$

where $D^{t_0}_{\boldsymbol{\pi}}(t)$ can be found based on the parameters obtained from the cost prediction module. The procedure is shown in Algorithm 5.1.

In Algorithm 5.1, whenever we solve (5.3), we get the value of $\boldsymbol{\pi}_{\text{off}}$ for additional T slots. Such a solution is sufficient in practice because we only need to know where to place the instances in the current slot. The value of $D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t)$ in (5.3) depends on the configuration in slot $t_0 - 1$, i.e. $\boldsymbol{\pi}(t_0 - 1)$, according to (5.1). When $t_0 = 1$, $\boldsymbol{\pi}(t_0 - 1)$ can be regarded as an arbitrary value, because the migration cost $W(t, \cdot, \cdot) = 0$ for $t = 1$.

The intuitive explanation of (5.3) is that, at the beginning of slot t_0, it finds the optimal configuration that minimizes the predicted cost over the next slots (including t_0) up to $t_0 + T - 1$, given the locations of instances in the previous slot $t_0 - 1$. We focus on solving (5.3) next.

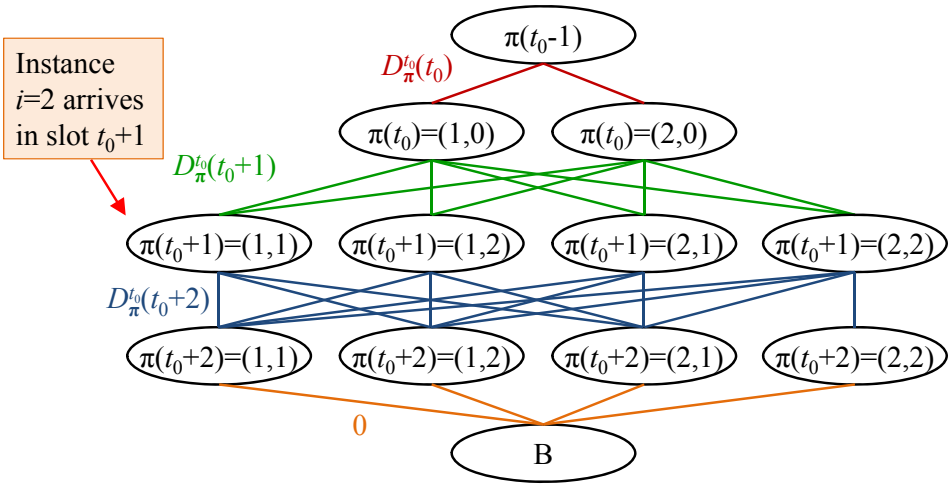


Figure 5.2: Shortest-path formulation with $N = 2$, $M = 2$, and $T = 3$. Instance $i = 1$ is running in all slots, instance $i = 2$ arrives at the beginning of slot $t_0 + 1$ and is running in slots $t_0 + 1$ and $t_0 + 2$.

5.3.2 Equivalence to Shortest-Path Problem

Problem (5.3) is equivalent to a shortest-path problem with $D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t)$ as weights, as shown in Fig. 5.2. Each edge represents one possible combination of configurations in adjacent timeslots, and the weight on each edge is the predicted cost for such configurations. The configuration in slot $t_0 - 1$ is always given, and the number of possible configurations in subsequent timeslots is at most N^M, where M is defined as in Section 5.2.1.2 for the current window $\{t_0, ..., t_0 + T - 1\}$. Node B is a dummy node to ensure that we find a single shortest path, and the edges connecting node B have zero weights. It is obvious that the optimal solution to (5.3) can be found by taking the shortest (minimum-weighted) path from node $\boldsymbol{\pi}(t_0 - 1)$ to node B in Fig. 5.2, and the nodes that the shortest path traverses correspond to the optimal solution $\boldsymbol{\pi}_{\text{off}}(t_0, ..., t_0 + T - 1)$ for (5.3).

5.3.3 Algorithm

We can solve the abovementioned shortest-path problem by means of dynamic programming [36]. The algorithm is shown in Algorithm 5.2, where we use $U_p(t, \mathbf{m})$

and $W_p(t, \tilde{\mathbf{m}}, \mathbf{m})$ to respectively denote the predicted local and migration costs, when $\boldsymbol{\pi}(t) = \mathbf{m}$ and $\boldsymbol{\pi}(t-1) = \tilde{\mathbf{m}}$. In the algorithm, Lines 5–16 iteratively finds the shortest path (minimum objective function) for each timeslot. In each iteration, the optimal solution for every possible (single-slot) configuration $\mathbf{m}$ is found by solving Bellman's equation of the problem (Line 11). Then, the final optimal configuration is found in Lines 17 and 18. It is obvious that output of this algorithm satisfies Bellman's principle of optimality, so the result is the shortest path and hence the optimal solution to (5.3).

Complexity: When the vectors $\boldsymbol{\pi}_{\mathbf{m}}$ and $\boldsymbol{\xi}_{\mathbf{m}}$ are stored as linked-lists, Algorithm 5.2 has time-complexity $O\left(N^{2M}T\right)$. This is because the minimization in Line 11 requires enumerating at most N^M possible configurations, and there can be at most $N^M T$ possible combinations of values of t and $\mathbf{m}$.

5.4 Complexity Reduction and Online Service Placement

The complexity of Algorithm 5.2 is exponential unless the number of service instances M is a constant. Therefore, it is desirable to reduce its complexity. In this section, we propose a method that can find an approximate solution to (5.3) and, at the same time, handle online service instance arrivals and departures. We will also show that (5.3) is NP-hard when the number of service instances M is non-constant, which justifies the need to solve (5.3) approximately in an efficient manner.

5.4.1 Procedure

In the online case, we modify the procedure given in Algorithm 5.1 so that instances are placed one-by-one, where each placement greedily minimizes the objective function given in (5.3), while the configurations of previously placed instances remain

Algorithm 5.2 Algorithm for solving (5.3)

Define variables $\mathbf{m}$ and $\tilde{\mathbf{m}}$ to represent configurations respectively in the current and previous iteration
Define vectors $\boldsymbol{\pi}_{\mathbf{m}}$ and $\boldsymbol{\xi}_{\mathbf{m}}$ for all $\mathbf{m}$, where $\boldsymbol{\pi}_{\mathbf{m}}$ (correspondingly, $\boldsymbol{\xi}_{\mathbf{m}}$) records the optimal configuration given that the configuration at the current (correspondingly, previous) timeslot of iteration is $\mathbf{m}$
Define variables $\mu_{\mathbf{m}}$ and $\nu_{\mathbf{m}}$ for all $\mathbf{m}$ to record the sum cost values from slot t_0 respectively to the current and previous slot of iteration, given that the configuration is $\mathbf{m}$ in the current or previous slot
Initialize $\mu_{\mathbf{m}} \leftarrow 0$ and $\boldsymbol{\pi}_{\mathbf{m}} \leftarrow \emptyset$ for all $\mathbf{m}$
for $t = t_0, ..., t_0 + T - 1$ **do**
  for all $\mathbf{m}$ **do**
    $\nu_{\mathbf{m}} \leftarrow \mu_{\mathbf{m}}$
    $\boldsymbol{\xi}_{\mathbf{m}} \leftarrow \boldsymbol{\pi}_{\mathbf{m}}$
  end for
  for all $\mathbf{m}$ **do**
    $\tilde{\mathbf{m}}^* \leftarrow \arg\min_{\tilde{\mathbf{m}}} \{\nu_{\tilde{\mathbf{m}}} + U_p(t, \mathbf{m}) + W_p(t, \tilde{\mathbf{m}}, \mathbf{m})\}$
    $\boldsymbol{\pi}_{\mathbf{m}}(t_0, ..., t-1) \leftarrow \boldsymbol{\xi}_{\tilde{\mathbf{m}}^*}(t_0, ..., t-1)$
    $\pi_{\mathbf{m}}(t) \leftarrow \mathbf{m}$
    $\mu_{\mathbf{m}} \leftarrow \nu_{\tilde{\mathbf{m}}^*} + U_p(t, \mathbf{m}) + W_p(t, \tilde{\mathbf{m}}^*, \mathbf{m})$
  end for
end for
$\mathbf{m}^* \leftarrow \arg\min_{\mathbf{m}} \mu_{\mathbf{m}}$
$\boldsymbol{\pi}_{\text{off}}(t_0, ..., t_0 + T - 1) \leftarrow \boldsymbol{\pi}_{\mathbf{m}^*}(t_0, ..., t_0 + T - 1)$
return $\boldsymbol{\pi}_{\text{off}}(t_0, ..., t_0 + T - 1)$

unchanged.

We assume that each service instance i has a maximum lifetime $T_{\text{life}}(i)$, denoting the maximum number of remaining timeslots (including the current slot) that the instance remains in the system. The value of $T_{\text{life}}(i)$ may be infinity for instances that can potentially stay in the system for an arbitrary amount of time. The actual time that the instance stays in the system may be shorter than $T_{\text{life}}(i)$, but it cannot be longer than $T_{\text{life}}(i)$. When an instance leaves the system before its maximum lifetime has elapsed, we say that such a service instance departure is *unpredictable*.

We denote the configuration matrix $\boldsymbol{\pi}$ following online placement by $\boldsymbol{\pi}_{\text{on}}$. The configuration $\boldsymbol{\pi}_{\text{on}}$ is updated every time when an instance arrives or unpredictably departs. At the beginning of the window (before any instance has arrived), it is initiated as an all-zero matrix.

For a specific look-ahead window $\{t_0, ..., t_0 + T - 1\}$, when service instance i arrives in slot $t \in \{t_0, ..., t_0 + T - 1\}$, we assume that this instance stays in the system until slot $t_e = \min\{t + T_{\text{life}}(i) - 1; t_0 + T - 1\}$, and accordingly update the configuration matrix by

$$\boldsymbol{\pi}_{\text{on}}(t, ..., t_e) = \arg \min_{\boldsymbol{\pi}(t_a,...,t_e)} \sum_{t=t_a}^{t_e} D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t) \tag{5.4}$$

$$s.t.\ \boldsymbol{\pi}(t, ..., t_e) = \boldsymbol{\pi}_{\text{on}}(t, ..., t_e) \text{ except for column } i.$$

Note that only the configuration of service instance i (which is stored in the ith column of $\boldsymbol{\pi}$) is found and updated in (5.4), the configurations of all other instances $i' \neq i$ remain unchanged. The solution to (5.4) can still be found with Algorithm 5.2. The only difference is that vectors $\mathbf{m}$ and $\tilde{\mathbf{m}}$ now become scalar values within $\{1, ..., N\}$, because we only consider the configuration of a single instance i. The complexity in this case becomes $O(N^2T)$. At the beginning of the window, all the instances that have not departed after slot $t_0 - 1$ are seen as arrivals in slot t_0, because

Algorithm 5.3 Procedure for online service placement

Initialize $t_0 = 1$
loop
  Initialize $\boldsymbol{\pi}_{\text{on}}(t_0, ..., t_0 + T - 1)$ as an all-zero matrix
  for each timeslot $t = t_0, ..., t_0 + T - 1$ **do**
    for each instance i arriving at the beginning of slot t **do**
      $t_e \leftarrow \min\{t + T_{\text{life}}(i) - 1; t_0 + T - 1\}$
      Update $\boldsymbol{\pi}_{\text{on}}(t, ..., t_e)$ with the result from (5.4)
      Apply configurations specified in the ith column of $\boldsymbol{\pi}_{\text{on}}(t, ..., t_e)$ for service instance i in timeslots $t, ..., t_e$ until instance i departs
    end for
    for each instance i departing at the end of slot t **do**
      Set the ith column of $\boldsymbol{\pi}_{\text{on}}(t + 1, ..., t_0 + T - 1)$ to zero
    end for
  end for
  $t_0 \leftarrow t_0 + T$
end loop

we independently consider the placements in each window of size T. When multiple instances arrive simultaneously, an arbitrary arrival sequence is assigned to them, and the instances are still placed one-by-one by greedily minimizing (5.4).

When an instance i unpredictably departs at the end of slot $t \in \{t_0, ..., t_0+T-1\}$, we update $\boldsymbol{\pi}_{\text{on}}$ such that the ith column of $\boldsymbol{\pi}_{\text{on}}(t + 1, ..., t_0 + T - 1)$ is set to zero.

The online procedure described above is shown in Algorithm 5.3. Recall that $\boldsymbol{\pi}_{\text{on}}(t, ..., t_e)$ and $\boldsymbol{\pi}_{\text{on}}(t + 1, ..., t_0 + T - 1)$ are both part of a larger configuration matrix $\boldsymbol{\pi}_{\text{on}}(t_0, ..., t_0 + T - 1)$ (see Section 5.2.1.2).

Complexity: When placing a total of M instances, for a specific look-ahead window with size T, we can find the configurations of these M instances with complexity $O(N^2TM)$, because (5.4) is solved M times, each with complexity $O(N^2T)$.

Remark: It is important to note that in the above procedure, the configuration matrix $\boldsymbol{\pi}_{\text{on}}$ (and thus the cost value $D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t)$ for any $t \in \{t_0, ..., t_0 + T - 1\}$) may vary upon instance arrival or departure. It follows that the T-slot sum cost $\sum_{t=t_0}^{t_0+T-1} D^{t_0}_{\boldsymbol{\pi}_{\text{on}}(t-1,t)}(t)$ may vary whenever an instance arrives or departs at an arbitrary slot $t \in \{t_0, ..., t_0 + T - 1\}$, and the value of $\sum_{t=t_0}^{t_0+T-1} D^{t_0}_{\boldsymbol{\pi}_{\text{on}}(t-1,t)}(t)$ stands

for the predicted sum cost (over the current window containing T slots) under the *current* configuration, assuming that no new instance arrives and no instance unpredictably departs in the future. This variation in configuration and cost upon instance arrival and departure is frequently noted in the performance analysis presented next.

5.4.2 Performance Analysis

It is clear that for a single look-ahead window, Algorithm 5.3 has polynomial time-complexity while Algorithm 5.1 has exponential time-complexity. In this subsection, we show the NP-hardness of the offline service placement problem, and discuss the optimality gap between the online algorithm and the optimal offline placement. Note that we only focus on a single look-ahead window in this subsection. The interplay of multiple look-ahead windows and the impact of the window size will be considered in Section 5.5.

5.4.2.1 Definitions

For simplicity, we analyze the performance for a slightly restricted (but still general) class of cost functions. We introduce some additional definitions next.

Indexing of Instances: Without loss of generality, we assume that the instance with lowest index in the current window $\{t_0, ..., t_0 + T - 1\}$ has index $i = 1$, and the last instance that arrives before the *current time of interest* has index $i = M$, where the current time of interest can be any time within the current window. With this definition, M does *not* need to be the largest index in window $\{t_0, ..., t_0 + T - 1\}$. Instead, it can be the index of *any* instance that arrives within $\{t_0, ..., t_0 + T - 1\}$ The cost of placing up to (and including) instance M is considered, where some instances $i \leq M$ may have already departed from the system.

Configuration Sequence: When considering a window of T slots, we define a set of T-dimensional vectors $\Lambda \triangleq \{(\lambda_1, ..., \lambda_T) : \lambda_m \in \{0, 1, ..., N\}, \forall m \in \{1, ..., T\},$ where λ_m is non-zero for at most one block of consecutive values of $m\}$. We also define a vector $\boldsymbol{\lambda} \in \Lambda$ to represent one possible *configuration sequence* of a single service instance across these T consecutive slots. For any instance i, the ith column of configuration matrix $\boldsymbol{\pi}(t_0, ..., t_0 + T - 1)$ is equal to one particular value of $\boldsymbol{\lambda}$.

We also define a binary variable $x_{i\boldsymbol{\lambda}}$, where $x_{i\boldsymbol{\lambda}} = 1$ if service instance i operates in configuration sequence $\boldsymbol{\lambda}$ across slots $\{t_0, ..., t_0 + T - 1\}$ (i.e., the ith column of $\boldsymbol{\pi}(t_0, ..., t_0 + T - 1)$ is equal to $\boldsymbol{\lambda}$), and $x_{i\boldsymbol{\lambda}} = 0$ otherwise. We always have $\sum_{\boldsymbol{\lambda} \in \Lambda} x_{i\boldsymbol{\lambda}} = 1$ for all $i \in \{1, ..., M\}$.

We note that the values of $x_{i\boldsymbol{\lambda}}$ may vary over time due to arrivals and unpredictable departures of service instances, which can be seen from Algorithm 5.3 and by noting the relationship between $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$. Before instance i arrives, $x_{i\boldsymbol{\lambda}_0} = 1$ for $\boldsymbol{\lambda}_0 = [0, ..., 0]$ which contains all zeros, and $x_{i\boldsymbol{\lambda}} = 0$ for $\boldsymbol{\lambda} \neq \boldsymbol{\lambda}_0$. Upon arrival of instance i, we have $x_{i\boldsymbol{\lambda}_0} = 0$ and $x_{i\boldsymbol{\lambda}_1} = 1$ for a particular $\boldsymbol{\lambda}_1$. When instance i unpredictably departs, its configuration sequence switches from $\boldsymbol{\lambda}_1$ to an alternative (but partly correlated) sequence $\boldsymbol{\lambda}_1'$ according to Line 11 in Algorithm 5.3, after which $x_{i\boldsymbol{\lambda}_1} = 0$ and $x_{i\boldsymbol{\lambda}_1'} = 1$.

Resource Consumption: We assume that the costs are related to the resource consumption, and for ease of presentation, we consider two types of resource consumptions. The first type is associated with serving user requests, i.e., data transmission and processing when a cloud is running a service instance, which we refer to as *local resource consumption*. The second type is associated with migration, i.e., migrating a service instance from one cloud to another cloud, which we refer to as *migration resource consumption*.

If we know that instance i operates under configuration sequence $\boldsymbol{\lambda}$, then we

know whether instance i is placed on cloud n in slot t, for any $n \in \{1, ..., N\}$ and $t \in \{t_0, ..., t_0+T-1\}$. We also know whether instance i is migrated from cloud n to cloud h ($h \in \{1, 2, ..., N\}$) between slots $t-1$ and t. We use $a_{i\boldsymbol{\lambda}n}(t) \geq 0$ to denote the local resource consumption at cloud n in slot t when instance i is operating under $\boldsymbol{\lambda}$, and use $b_{i\boldsymbol{\lambda}nh}(t) \geq 0$ to denote the migration resource consumption when instance i operating under $\boldsymbol{\lambda}$ is assigned to cloud n in slot $t-1$ and to cloud h in slot t, where we note that the configuration in slot $t_0 - 1$ (before the start of the current window) is assumed to be given and thus independent of $\boldsymbol{\lambda}$. The values of $a_{i\boldsymbol{\lambda}n}(t)$ and $b_{i\boldsymbol{\lambda}nh}(t)$ are either service-specific parameters that are known beforehand, or they can be found as part of cost prediction.

We denote the sum local resource consumption at cloud n by $y_n(t) \triangleq \sum_{i=1}^{M} \sum_{\boldsymbol{\lambda}\in\Lambda} a_{i\boldsymbol{\lambda}n}(t) x_{i\boldsymbol{\lambda}}$, and denote the sum migration resource consumption from cloud n to cloud h by $z_{nh}(t) \triangleq \sum_{i=1}^{M} \sum_{\boldsymbol{\lambda}\in\Lambda} b_{i\boldsymbol{\lambda}nh}(t) x_{i\boldsymbol{\lambda}}$. We may omit the argument t in the following discussion.

Remark: The local and migration resource consumptions defined above can be related to CPU occupation, communication bandwidth, etc., or the sum of them. We only consider these two types of resource consumption for ease of presentation. By applying the same theoretical framework, the performance gap results (presented later) can be extended to incorporate multiple types of resources and more sophisticated cost functions, and similar results are obtained for the general case.

Costs: We refine the costs defined in Section 5.2.1.3 by considering the cost for each cloud or each pair of clouds. The local cost at cloud n in timeslot t is denoted by $u_{n,t}\left(y_n(t)\right)$. When a service instance is initiated in slot t, the local cost in slot t also includes the cost of initial placement of the corresponding service instance(s). The migration cost from cloud n to cloud h between slots $t-1$ and t is denoted by $w_{nh,t}\left(y_n(t-1), y_h(t), z_{nh}(t)\right)$. Besides $z_{nh}(t)$, the migration cost is also related

to $y_n(t-1)$ and $y_h(t)$, because additional processing may be needed for migration, and the cost for such processing can be related to the current load at clouds n and h. The functions $u_{n,t}(y)$ and $w_{nh,t}(y_n, y_h, z_{nh})$ can be different for different timeslots t and different clouds n and h, and they can depend on many factors, such as user location, network condition, background load of the cloud, etc. Noting that any constant term added to the cost function does not affect the optimal configuration, we set $u_{n,t}(0) = 0$ and $w_{nh,t}(0,0,0) = 0$. We also set $w_{nh,t}(\cdot,\cdot,0) = 0$, because there is no migration cost if we do not migrate. There is also no migration cost at the start of the first timeslot, thus we set $w_{nh,t}(\cdot,\cdot,\cdot) = 0$ for $t = 1$. With the above definition, the aggregated costs $U(t, \boldsymbol{\pi}(t))$ and $W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t))$ can be explicitly expressed as

$$U(t, \boldsymbol{\pi}(t)) \triangleq \sum_{n=1}^{N} u_{n,t}(y_n(t)) \tag{5.5}$$

$$W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t)) \triangleq \sum_{n=1}^{N}\sum_{h=1}^{N} w_{nh,t}(y_n(t-1), y_h(t), z_{nh}(t)) \,. \tag{5.6}$$

We then assume that the following assumption is satisfied for the cost functions, which holds for a large class of practical cost functions, such as those related to delay performance or load balancing [17].

Assumption 5.1. *Both $u_{n,t}(y)$ and $w_{nh,t}(y_n, y_h, z_{nh})$ are* convex non-decreasing *functions of y (or y_n, y_h, z_{nh}), satisfying:*

- $\frac{du_{n,t}}{dy}(0) > 0$
- $\frac{\partial w_{nh,t}}{\partial z_{nh}}(\cdot,\cdot,0) > 0$ *for* $t \geq 2$

for all t, n, and h (unless stated otherwise), where $\frac{du_{n,t}}{dy}(0)$ denotes the derivative of $u_{n,t}$ with respect to (w.r.t.) y evaluated at $y = 0$, and $\frac{\partial w_{nh,t}}{\partial z_{nh}}(\cdot,\cdot,0)$ denotes the partial derivative of $w_{nh,t}$ w.r.t. z_{nh} evaluated at $z_{nh} = 0$ and arbitrary y_n and y_h.

Vector Notation: To simplify the presentation, we use vectors to denote a collection of variables across multiple clouds, slots, or configuration sequences. For simplicity, we index each element in the vector with multiple indices that are related to the index of the element, and use the general notion $(\mathbf{g})_{m_1 m_2}$ (or $(\mathbf{g})_{m_1 m_2 m_3}$) to denote the (m_1, m_2)th (or (m_1, m_2, m_3)th) element in an arbitrary vector $\mathbf{g}$. Because we know the range of each index, multiple indices can be easily mapped to a single index (such as using the vectorization operation when $\mathbf{g}$ is a matrix). We regard each vector as a *single-indexed* vector for the purpose of vector concatenation (i.e., joining two vectors into one vector) and gradient computation later.

We define vectors $\mathbf{y}$ (with NT elements), $\mathbf{z}$ (with N^2T elements), $\mathbf{x}$ (with $M(N+1)^T$ elements), $\mathbf{a}_{i\boldsymbol{\lambda}}$ (with NT elements), and $\mathbf{b}_{i\boldsymbol{\lambda}}$ (with N^2T elements), for every value of $i \in \{1, 2, ..., M\}$ and $\boldsymbol{\lambda} \in \Lambda$. Different values of i and $\boldsymbol{\lambda}$ correspond to different vectors $\mathbf{a}_{i\boldsymbol{\lambda}}$ and $\mathbf{b}_{i\boldsymbol{\lambda}}$. The elements in these vectors are defined as follows:

$$(\mathbf{y})_{nt} \triangleq y_n(t),\ (\mathbf{z})_{nht} \triangleq z_{nh}(t),\ (\mathbf{x})_{i\boldsymbol{\lambda}} \triangleq x_{i\boldsymbol{\lambda}},$$

$$(\mathbf{a}_{i\boldsymbol{\lambda}})_{nt} \triangleq a_{i\boldsymbol{\lambda}n}(t),\ (\mathbf{b}_{i\boldsymbol{\lambda}})_{nht} \triangleq b_{i\boldsymbol{\lambda}nh}(t).$$

As discussed earlier in this section, $x_{i\boldsymbol{\lambda}}$ may unpredictably change over time due to arrivals and departures of service instances. It follows that the vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ may vary over time (recall that $\mathbf{y}$ and $\mathbf{z}$ are dependent on $\mathbf{x}$ by definition). The vectors $\mathbf{a}_{i\boldsymbol{\lambda}}$ and $\mathbf{b}_{i\boldsymbol{\lambda}}$ are constant.

Alternative Cost Expression: Using the above definitions, we can write the sum cost of all T slots as follows

$$\widetilde{D}(\mathbf{x}) \triangleq \widetilde{D}(\mathbf{y}, \mathbf{z}) \triangleq \sum_{t=t_0}^{t_0+T-1} \left[\sum_{n=1}^{N} u_{n,t}(y_n(t)) + \sum_{n=1}^{N} \sum_{h=1}^{N} w_{nh,t}(y_n(t-1), y_h(t), z_{nh}(t)) \right] \tag{5.7}$$

where the cost function $\widetilde{D}(\cdot)$ can be expressed either in terms of $\mathbf{x}$ or in terms of $(\mathbf{y}, \mathbf{z})$. The cost function defined in (5.7) is equivalent to $\sum_{t=t_0}^{t_0+T-1} D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t)$, readers are also referred to the per-slot cost defined in (5.1) for comparison. The value of $\widetilde{D}(\mathbf{x})$ or, equivalently, $\widetilde{D}(\mathbf{y}, \mathbf{z})$ may vary over time due to service arrivals and unpredictable service instance departures as discussed above.

5.4.2.2 Equivalent Problem Formulation

With the above definitions, the offline service placement problem in (5.3) can be equivalently formulated as the following, where our goal is to find the optimal configuration for all service instances $1, 2, ..., M$ (we consider the offline case here where we know when each instance arrives and no instance will unpredictably leave after they have been placed):

$$\begin{aligned} \min_{\mathbf{x}} \quad & \widetilde{D}(\mathbf{x}) \\ s.t. \quad & \sum_{\boldsymbol{\lambda} \in \Lambda_i} x_{i\boldsymbol{\lambda}} = 1, \forall i \in \{1, 2, ..., M\} \\ & x_{i\boldsymbol{\lambda}} \in \{0, 1\}, \forall i \in \{1, 2, ..., M\}, \boldsymbol{\lambda} \in \Lambda_i \end{aligned} \tag{5.8}$$

where $\Lambda_i \subseteq \Lambda$ is a subset of configuration sequences that contains those vectors whose elements are non-zero starting from the slot at which i arrives and ending at the slot at which i departs, while all other elements of the vectors are zero.

We now show that (5.8), and thus (5.3), is NP-hard even in the offline case, which further justifies the need for an approximation algorithm for solving the problem.

Proposition 5.1. ***(NP-Hardness)*** *The problem in (5.8), and thus (5.3), is NP-hard.*

Proof. We show that problem (5.8) can be reduced from the partition problem, which is known to be NP-complete [55, Corollary 15.28]. The partition problem is defined as follows.

Definition of Partition Problem: Given *positive integers* $v_1, v_2, ..., v_M$, is there a subset $\mathcal{S} \subseteq \{1, 2, ..., M\}$ such that $\sum_{j\in\mathcal{S}} v_j = \sum_{j\in\mathcal{S}^c} v_j$, where $\mathcal{S}^c$ is the complement set of $\mathcal{S}$?

Similar to the proof of [55, Theorem 18.1], we define a decision version of the bin packing problem, where we assume that there are M items each with size

$$a_i \triangleq \frac{2v_i}{\sum_{j=1}^{M} v_j}$$

for all $i \in \{1, 2, ..., M\}$, and the problem is to determine whether these M items can be packed into *two bins* each with *unit size* (i.e., its size is equal to one). It is obvious that this bin packing decision problem is equivalent to the partition problem.

To solve the above defined bin packing decision problem, we can set $t_0 = 1$, $T = 1$, and $N = 2$ in (5.8). Because we attempt to place all items, we set $\Lambda_i = \{1, 2\}$ for all i. By definition, $w_{kl,t}(\cdot, \cdot, \cdot) = 0$ for $t = 1$. We omit the subscript t in the following as we only consider a single slot. We define $a_{i\boldsymbol{\lambda}n} = a_i$ for all $\boldsymbol{\lambda}, n$, and define

$$u_n(y) = \begin{cases} \varepsilon y, & \text{if } y \leq 1 \\ \frac{2\varepsilon}{c}(y-1) + \varepsilon, & \text{if } y > 1 \end{cases} \tag{5.9}$$

where $c \triangleq \frac{1}{\sum_{j=1}^{M} v_j}$, and $\varepsilon > 0$ is an arbitrary constant.

Because v_i is a positive integer for any i, we have that $\frac{a_i}{c} = 2v_i$ is always a positive integer, and $\frac{1}{c} = \sum_{j=1}^{M} v_j$ is also always a positive integer. It follows that y can only be integer multiples of c (where we recall that y is the sum of a_i for those items i that are placed in the bin), and there exists a positive integer $c' \triangleq \sum_{j=1}^{M} v_j$ such that $c'c = 1$. Thus, when $y > 1$, we always have $y - 1 \geq c$. Therefore, the choice of $u_n(y)$ in (5.9) guarantees that $u_n(y_n) \geq 3\varepsilon > 2\varepsilon$ whenever bin n ($n \in \{1, 2\}$) exceeds its size, and $\sum_{n=1}^{2} u_n(y_n) \leq 2\varepsilon$ when no bin has exceeded its size. At the same time, $u_n(y)$ satisfies Assumption 5.1 as long as $c \leq 2$.

By the definition of c, we always have $c \leq 2$ because $\sum_{j=1}^{M} v_j \geq 1$. To solve the bin packing decision problem defined above (thus the partition problem), we can solve (5.8) with the above definitions. If the solution is not larger than 2ε, the packing is feasible and the answer to the partition problem is "yes"; otherwise, the packing is infeasible and the answer to the partition problem is "no". It follows that problem (5.8) is "at least as hard as" the partition problem, which proves that (5.8) is NP-hard. □

An online version of problem (5.8) can be constructed by updating Λ_i over time. When an arbitrary instance i has not yet arrived, we define Λ_i as the set containing an all-zero vector. After instance i arrives, we assume that it will run in the system until t_e (defined in Section 5.4.1), and update Λ_i to conform to the arrival and departure times of instance i (see above). After instance i departs, Λ_i can be further updated so that the configurations corresponding to all remaining slots are zero.

5.4.2.3 Performance Gap

As discussed earlier, Algorithm 5.3 solves (5.8) in a greedy manner, where each service instance i is placed to greedily minimize the objective function in (5.8). In the following, we compare the result from Algorithm 5.3 with the true optimal result, where the optimal result assumes offline placement. We use $\mathbf{x}$ and $(\mathbf{y}, \mathbf{z})$ to denote the result from Algorithm 5.3, and use $\mathbf{x}^*$ and $(\mathbf{y}^*, \mathbf{z}^*)$ to denote the optimal result.

Lemma 5.1. ***(Convexity of** $\widetilde{D}(\cdot)$**)*** *When Assumption 5.1 is satisfied, the cost function* $\widetilde{D}(\mathbf{x})$ *or, equivalently,* $\widetilde{D}(\mathbf{y}, \mathbf{z})$ *is a non-decreasing convex function w.r.t.* $\mathbf{x}$*, and it is also a non-decreasing convex function w.r.t.* $\mathbf{y}$ *and* $\mathbf{z}$*.*

Proof. According to Assumption 5.1, $u_{n,t}(y_n(t))$ and $w_{nh,t}(y_n(t-1), y_h(t), z_{nh}(t))$ are non-decreasing convex functions. Because $y_n(t)$ and $z_{nh}(t)$ are linear mappings of $x_{i\boldsymbol{\lambda}}$ with non-negative weights for any t, n, and h, and also because the sum

of non-decreasing convex functions is still a non-decreasing convex function, the lemma holds [56, Section 3.2]. □

In the following, we use $\nabla_{\mathbf{x}}$ to denote the gradient w.r.t. *each element* in vector $\mathbf{x}$, i.e., the $(i, \boldsymbol{\lambda})$th element of $\nabla_{\mathbf{x}}\widetilde{D}(\mathbf{x})$ is $\frac{\partial \widetilde{D}(\mathbf{x})}{\partial x_{i\boldsymbol{\lambda}}}$. Similarly, we use $\nabla_{\mathbf{y},\mathbf{z}}$ to denote the gradient w.r.t. each element in vector $(\mathbf{y}, \mathbf{z})$, where $(\mathbf{y}, \mathbf{z})$ is a vector that concatenates vectors $\mathbf{y}$ and $\mathbf{z}$.

Proposition 5.2. ***(Performance Gap)*** *When Assumption 5.1 is satisfied, we have*

$$\widetilde{D}(\mathbf{x}) \leq \widetilde{D}(\phi\psi\mathbf{x}^*) \tag{5.10}$$

or, equivalently,

$$\widetilde{D}\left(\mathbf{y}, \mathbf{z}\right) \leq \widetilde{D}\left(\phi\psi\mathbf{y}^*, \phi\psi\mathbf{z}^*\right) \tag{5.11}$$

where ϕ and ψ are constants satisfying

$$\phi \geq \frac{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left(\mathbf{y}_{max} + \mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{z}_{max} + \mathbf{b}_{i\boldsymbol{\lambda}}\right) \cdot \left(\mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{b}_{i\boldsymbol{\lambda}}\right)}{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left(\mathbf{y}, \mathbf{z}\right) \cdot \left(\mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{b}_{i\boldsymbol{\lambda}}\right)} \tag{5.12}$$

$$\psi \geq \frac{\nabla_{\mathbf{x}}\widetilde{D}\left(\mathbf{x}\right) \cdot \mathbf{x}}{\widetilde{D}(\mathbf{x})} = \frac{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left(\mathbf{y}, \mathbf{z}\right) \cdot \left(\mathbf{y}, \mathbf{z}\right)}{\widetilde{D}\left(\mathbf{y}, \mathbf{z}\right)} \tag{5.13}$$

for any i and $\boldsymbol{\lambda} \in \Lambda_i$, in which $\mathbf{y}_{max}$ and $\mathbf{z}_{max}$ respectively denote the maximum values of $\mathbf{y}$ and $\mathbf{z}$ (the maximum is taken element-wise) at any time *within slots $\{t_0, ..., t_0 + T - 1\}$ until the current time of interest, $(\mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{b}_{i\boldsymbol{\lambda}})$ is a vector that concatenates $\mathbf{a}_{i\boldsymbol{\lambda}}$ and $\mathbf{b}_{i\boldsymbol{\lambda}}$, and "$\cdot$" denotes the dot-product.*

Proof. See Appendix C.4. □

Remark: We note that according to the definition of M in Section 5.4.2.1, the bound given in Proposition 5.2 holds at any time of interest within slots $\{t_0, ..., t_0 + T - 1\}$, i.e., for any number of instances that has arrived to the system, where some of them may have already departed.

5.4.2.4 Intuitive Explanation to the Constants ϕ and ψ

The constants ϕ and ψ in Proposition 5.2 are related to "how convex" the cost function is. Here, we use the second order derivative as a measure of convexity, and we say that a function is *more convex* if it has a larger second order derivative. In other words, they are related to how fast the cost of placing a single instance changes under different amount of existing resource consumption.

Figure 5.3 shows an illustrative example, where we only consider one cloud and one timeslot (i.e., $t = 1$, $T = 1$, and $N = 1$). In this case, setting $\phi = \frac{d\widetilde{D}}{dy}(y_{\max} + a_{\max}) \big/ \frac{d\widetilde{D}}{dy}(y)$ satisfies (5.12), where $a_{\max}$ denotes the maximum resource consumption of a single instance. Similarly, setting $\psi = \frac{d\widetilde{D}}{dy}(y) \cdot y \big/ \widetilde{D}(y)$ satisfies (5.13). We can see that the values of ϕ and ψ need to be larger when the cost function is more convex. For the general case, there is a weighted sum in both the numerator and denominator in (5.12) and (5.13). However, when we look at a single cloud (for the local cost) or a single pair of clouds (for the migration cost) in a single timeslot, the above intuition still applies.

So, why is the optimality gap larger when the cost functions are more convex, i.e., have a larger second order derivative? We note that in the greedy assignment procedure in Algorithm 5.3, we choose the configuration of each instance i by minimizing the cost under the system state at the time when instance i arrives, where the system state represents the local and migration resource consumptions as specified by vectors $\mathbf{y}$ and $\mathbf{z}$. When cost functions are more convex, for an alternative system state $(\mathbf{y}', \mathbf{z}')$, it is more likely that the placement of instance i (which was determined at system state $(\mathbf{y}, \mathbf{z})$) becomes far from optimum. This is because if cost functions are more convex, the cost increase of placing a new instance i (assuming the same configuration for i) varies more when $(\mathbf{y}, \mathbf{z})$ changes. This intuition is also confirmed by formal results as described next.

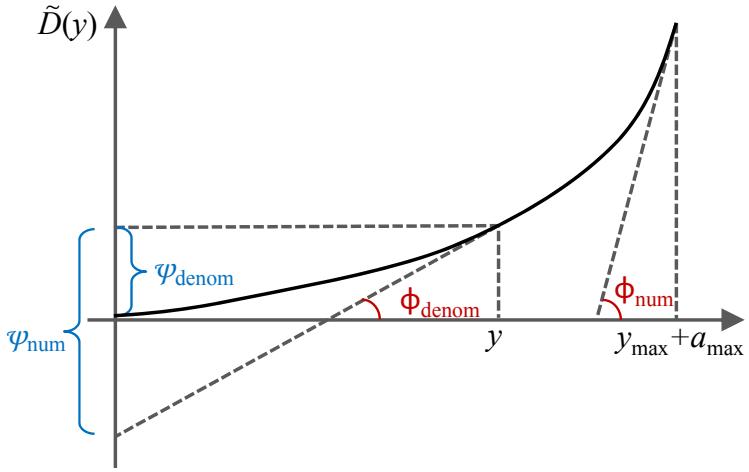


Figure 5.3: Illustration of the performance gap for $t = 1$, $T = 1$, and $N = 1$, where $a_{\max}$ denotes the maximum resource consumption of a single instance. In this example, (5.12) becomes $\phi \geq \frac{\phi_{\text{num}}}{\phi_{\text{denom}}}$, and (5.13) becomes $\psi \geq \frac{\psi_{\text{num}}}{\psi_{\text{denom}}}$.

5.4.2.5 Performance with Linear Cost Functions

When the cost functions are linear, in the form of

$$u_{n,t}(y) = \gamma_{n,t} y \tag{5.14}$$

$$w_{nh,t}\left(y_n, y_h, z_{nh}\right) = \kappa^{(1)}_{nh,t} y_n + \kappa^{(2)}_{nh,t} y_h + \kappa^{(3)}_{nh,t} z_{nh} \tag{5.15}$$

where the constants $\gamma_{n,t} \geq 0$ and $\kappa^{(1)}_{nh,t}, \kappa^{(2)}_{nh,t}, \kappa^{(3)}_{nh,t} \geq 0$, we have the following result.

Proposition 5.3. *When the cost functions are defined as in (5.14) and (5.15) while satisfying Assumption 5.1, Algorithm 5.3 provides the optimal solution.*

Proof. We have

$$\nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}\left(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{z}_{\max} + \mathbf{b}_{i\boldsymbol{\lambda}}\right) = \nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}\left(\mathbf{y}, \mathbf{z}\right)$$

$$\nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}\left(\mathbf{y}, \mathbf{z}\right) \cdot \left(\mathbf{y}, \mathbf{z}\right) = \widetilde{D}\left(\mathbf{y}, \mathbf{z}\right)$$

because the gradient in this case is a constant. Hence, choosing $\phi = \psi = 1$ satisfies (5.12) and (5.13), yielding $\widetilde{D}(\mathbf{x}) \leq \widetilde{D}(\mathbf{x}^*)$ which means that the solution from

Algorithm 5.3 is not worse than the optimal solution. □

This implies that the greedy service placement is optimal for linear cost functions, which is intuitive because the previous placements have no impact on the cost of later placements when the cost function is linear.

5.4.2.6 Performance with Polynomial Cost Functions

Consider polynomial cost functions in the form of

$$u_{n,t}(y) = \sum_{\rho} \gamma_{n,t}^{(\rho)} y^{\rho} \tag{5.16}$$

$$w_{nh,t}\left(y_n, y_h, z_{nh}\right) = \sum_{\rho_1} \sum_{\rho_2} \sum_{\rho_3} \kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} y_n^{\rho_1} y_h^{\rho_2} z_{nh}^{\rho_3} \tag{5.17}$$

where $\rho, \rho_1, \rho_2, \rho_3$ are integers satisfying $\rho \geq 1$, $\rho_1 + \rho_2 + \rho_3 \geq 1$ and the constants $\gamma_{n,t}^{(\rho)} \geq 0$, $\kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} \geq 0$.

We first introduce the following assumption which can be satisfied in most practical systems with an upper bound on resource consumptions and departure rates.

Assumption 5.2. *The following is satisfied:*

- *For all $i, \boldsymbol{\lambda}, n, h, t$, there exists a constants a_{max} and b_{max}, such that*

$$a_{i\boldsymbol{\lambda}n}(t) \leq a_{max} \text{ and } b_{i\boldsymbol{\lambda}nh}(t) \leq b_{max}$$

- *The number of instances that unpredictably leave the system in each slot is upper bounded by a constant B_d.*

Proposition 5.4. *Assume that the cost functions are defined as in (5.16) and (5.17) while satisfying Assumptions 5.1 and 5.2.*

Let Ω denote the highest order of the polynomial cost functions. Specifically, $\Omega \triangleq \max\{\rho; \rho_1 + \rho_2 + \rho_3\}$, subject to $\gamma_{n,t}^{(\rho)} > 0$ and $\kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} > 0$.

Define $\Gamma(\mathcal{I}(M)) \triangleq \widetilde{D}(\mathbf{x}_{\mathcal{I}(M)}) \Big/ \widetilde{D}(\mathbf{x}^*_{\mathcal{I}(M)})$*, where* $\mathcal{I}(M)$ *is a problem input*[3] *containing* M *instances, and* $\mathbf{x}_{\mathcal{I}(M)}$ *and* $\mathbf{x}^*_{\mathcal{I}(M)}$ *are respectively the online and offline (optimal) results for input* $\mathcal{I}(M)$*. We say that Algorithm 5.3 is* c*-competitive in placing* M *instances if* $\Gamma \triangleq \max_{\mathcal{I}(M)} \Gamma(\mathcal{I}(M)) \leq c$ *for a given* M*.*

We have:

- *Algorithm 5.3 is* $O(1)$*-competitive.*
- *In particular, for any* $\delta > 0$*, there exists a sufficiently large* M*, such that Algorithm 5.3 is* $\left(\Omega^{\Omega} + \delta\right)$*-competitive.*

Proof. See Appendix C.5. □

Proposition 5.4 states that the competitive ratio does not indefinitely increase with increasing number of instances (specified by M). Instead, it approaches a constant value when M becomes large.

When the cost functions are linear as in (5.14) and (5.15), we have $\Omega = 1$. In this case, Proposition 5.4 gives a competitive ratio upper bound of $1 + \delta$ (for sufficiently large M) where $\delta > 0$ can be arbitrarily small, while Proposition 5.3 shows that Algorithm 5.3 is optimal. This means that the competitive ratio upper bound given in Proposition 5.4 is *asymptotically tight* as M goes to infinity.

5.4.2.7 Performance with Other Forms of Cost Functions

Algorithm 5.3 is also $O(1)$-competitive for some more general forms of cost functions. For example, consider a simple case where there is no migration resource consumption, i.e. $b_{i\boldsymbol{\lambda}nh}(t) = 0$ for all $i, \boldsymbol{\lambda}, n, h$. Define $u_{n_0,t}(y) = \gamma y$ for some cloud n_0 and all t, where $\gamma > 0$ is a constant. For all other clouds $n \neq n_0$, define $u_{n,t}(y)$ as a general cost function while satisfying Assumption 5.1 and some additional mild

[3] A particular problem input specifies the time each instance arrives/departs as well as the values of $\mathbf{a}_{i\boldsymbol{\lambda}}$ and $\mathbf{b}_{i\boldsymbol{\lambda}}$ for each $i, \boldsymbol{\lambda}$.

assumptions presented below. Assume that there exists a constant $a_{\max}$ such that $a_{i\boldsymbol{\lambda}n}(t) \leq a_{\max}$ for all $i, \boldsymbol{\lambda}, n, t$.

Because $u_{n,t}(y)$ is convex non-decreasing and Algorithm 5.3 operates in a greedy manner, if $\frac{du_{n,t}}{dy}(y) > \gamma$, no new instance will be placed on cloud n, as it incurs higher cost than placing it on n_0. As a result, the maximum value of $y_n(t)$ is bounded, let us denote this upper bound by $y_n^{\max}(t)$. We note that $y_n^{\max}(t)$ is *only dependent on the cost function definition* and is independent of the number of arrived instances.

Assume $u_{n,t}(y_n^{\max}(t)) < \infty$ and $\frac{du_{n,t}}{dy}(y_n^{\max}(t) + a_{\max}) < \infty$ for all $n \neq n_0$ and t. When ignoring the cost at cloud n_0, the ratio $\Gamma(\mathcal{I}(M))$ does not indefinitely grow with incoming instances, because among all $y_n(t) \in [0, y_n^{\max}(t)]$ for all t and $n \neq n_0$, we can find ϕ and ψ that satisfy (5.12) and (5.13), we can also find the competitive ratio $\Gamma \triangleq \max_{\mathcal{I}(M)} \Gamma(\mathcal{I}(M))$. The resulting Γ is only dependent on the cost function definition, hence it does not keep increasing with M. Taking into account the cost at cloud n_0, the above result still applies, because the cost at n_0 is linear in $y_{n_0}(t)$, so that in either of (5.12), (5.13), or in the expression of $\Gamma(\mathcal{I}(M))$, the existence of this linear cost only adds a same quantity (which might be different in different expressions though) to both the numerator and denominator, which does not increase Γ (because $\Gamma \geq 1$).

The cloud n_0 can be considered as the backend cloud, which usually has abundant resources thus its cost-per-unit-resource often remains unchanged. This example can be generalized to cases with non-zero migration resource consumption, and we will illustrate such an application in the simulations in Section 5.6.

5.5 Optimal Look-Ahead Window Size

In this section, we study how to find the optimal window size T to look-ahead. When there are no errors in the cost prediction, setting T as large as possible can

potentially bring the best long-term performance. However, the problem becomes more complicated when we consider the prediction error, because the farther ahead we look into the future, the less accurate the prediction becomes. When T is large, the predicted cost value may be far away from the actual cost, which can cause the configuration obtained from the predicted cost with size-T windows (denoted by $\boldsymbol{\pi}_p$) deviate significantly from the true optimal configuration (obtained from actual costs) $\boldsymbol{\pi}^*$. Note that $\boldsymbol{\pi}_p$ and $\boldsymbol{\pi}^*$ specify the configurations for an arbitrarily large number of timeslots, as in (5.2). Conversely, when T is small, the solution may not perform well in the long-term, because the look-ahead window is small and the long-term effect of service placement is not considered. Therefore, we have to find the optimal value of T which minimizes both the impact of prediction error and the impact of truncating the look-ahead time-span.

We first define a constant σ, which satisfies

$$\max_{\boldsymbol{\pi}(t-1,t)} W_a(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t)) \leq \sigma \tag{5.18}$$

for any t, to represent the maximum value of the actual migration cost in any slot, where $W_a(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t))$ denotes the actual migration cost. The value of σ is system-specific and is related to the cost definition.

To help with our analysis below, we define the sum-error starting from slot t_0 up to slot $t_0 + T - 1$ as

$$F(T) \triangleq \sum_{t=t_0}^{t_0+T-1} \varepsilon(t - t_0). \tag{5.19}$$

Because $\varepsilon(t - t_0) \geq 0$ and $\varepsilon(t - t_0)$ is non-decreasing with t, it is obvious that $F(T+2) - F(T+1) \geq F(T+1) - F(T)$. Hence, $F(T)$ is a convex non-decreasing function for $T \geq 0$, where we define $F(0) = 0$.

5.5.1 Upper Bound on Cost Difference

In the following, we focus on the objective function given in (5.2), and study how worse the configuration $\boldsymbol{\pi}_p$ can perform, compared to the optimal configuration $\boldsymbol{\pi}^*$.

Proposition 5.5. *For look-ahead window size T, suppose that we can solve (5.3) with competitive ratio $\Gamma \geq 1$, the upper bound on the cost difference (while taking the competitive ratio Γ into account) from placement sequences $\boldsymbol{\pi}_p$ and $\boldsymbol{\pi}^*$ is given by*

$$\lim_{T_{max}\to\infty}\left(\frac{\sum_{t=1}^{T_{max}} A_{\boldsymbol{\pi}_p}(t)}{T_{max}} - \Gamma\frac{\sum_{t=1}^{T_{max}} A_{\boldsymbol{\pi}^*}(t)}{T_{max}}\right) \leq \frac{(\Gamma+1)F(T)+\sigma}{T}. \tag{5.20}$$

Proof. Define $T_{\max} > 1$ as an arbitrarily large timeslot index. We note that there are $\left\lfloor \frac{T_{\max}}{T} \right\rfloor$ *full* look-ahead windows of size T within timeslots from 1 to $T_{\max}$, where $\lfloor x \rfloor$ denotes the integral part of x. In the last window, there are $T_{\max} - T \cdot \left\lfloor \frac{T_{\max}}{T} \right\rfloor$ slots. We have

$$F\left(T_{\max} - T \cdot \left\lfloor \frac{T_{\max}}{T} \right\rfloor\right) \leq \frac{T_{\max} - T \cdot \left\lfloor \frac{T_{\max}}{T} \right\rfloor}{T} F(T) \tag{5.21}$$

because $F(T)$ is convex non-decreasing and $F(0) = 0$.

For the true optimal configuration $\boldsymbol{\pi}^*$, according to the definitions of $\varepsilon(\tau)$ and $F(T)$, the difference in the predicted and actual sum-costs satisfies

$$\begin{aligned}
&\sum_{t=1}^{T_{\max}} D_{\boldsymbol{\pi}^*}(t) - \sum_{t=1}^{T_{\max}} A_{\boldsymbol{\pi}^*}(t) \\
&\leq \left\lfloor \frac{T_{\max}}{T} \right\rfloor F(T) + F\left(T_{\max} - T \cdot \left\lfloor \frac{T_{\max}}{T} \right\rfloor\right) \\
&\leq \frac{T_{\max}}{T} F(T)
\end{aligned} \tag{5.22}$$

where the last inequality follows from (5.21). Similarly, for the configuration $\boldsymbol{\pi}_p$ obtained from predicted costs, we have

$$\sum_{t=1}^{T_{\max}} A_{\boldsymbol{\pi}_p}(t) - \sum_{t=1}^{T_{\max}} D_{\boldsymbol{\pi}_p}(t) \leq \frac{T_{\max}}{T} F(T). \tag{5.23}$$

In the following, we establish the relationship between $\boldsymbol{\pi}^*$ and $\boldsymbol{\pi}_p$. Assume that, in (5.3), we neglect the migration cost at the beginning of each look-ahead window, i.e. we consider each window independently and there is no migration cost in the first timeslot of each window, then we have

$$\sum_{t=1}^{T_{\max}} D_{\boldsymbol{\pi}_p}(t) \leq \Gamma \sum_{t=1}^{T_{\max}} D_{\boldsymbol{\pi}^*}(t)$$

where the constant $\Gamma \geq 1$ is the competitive ratio of solving (5.3). This holds because there is no connection between different windows, thus the optimal sequences (considering predicted costs) obtained from (5.3) constitute the optimal sequence up to a factor Γ for all timeslots $[1, T_{\max}]$. Now we relax the assumption and consider the existence of migration cost in the first slot of each window. Note that we cannot have more than $\left\lfloor \frac{T_{\max}}{T} \right\rfloor + 1$ windows and the first timeslot $t = 1$ does not have migration cost. Thus,

$$\sum_{t=1}^{T_{\max}} D_{\boldsymbol{\pi}_p}(t) \leq \Gamma \sum_{t=1}^{T_{\max}} D_{\boldsymbol{\pi}^*}(t) + \frac{T_{\max}}{T}\sigma. \tag{5.24}$$

The bound holds because regardless of the configuration in slot $t_0 - 1$, the migration cost in slot t_0 cannot exceed σ.

By multiplying Γ on both sides of (5.22) and summing up the result with (5.24), we get

$$\sum_{t=1}^{T_{\max}} D_{\boldsymbol{\pi}_p}(t) - \Gamma \sum_{t=1}^{T_{\max}} A_{\boldsymbol{\pi}^*}(t) \leq \frac{T_{\max}}{T} \left(\Gamma F(T) + \sigma\right). \tag{5.25}$$

Summing up (5.23) with (5.25), dividing both sides by $T_{\max}$, and taking the limit on both sides yields the proposition. □

We assume in the following that the competitive ratio Γ is independent of the choice of T, and regard it as a given parameter in the problem of finding optimal T. This assumption is justified because the performance gap results in Propositions 5.2, 5.3, and 5.4 are not related to the window size T. We define the optimal look-ahead

window size as the solution to the following optimization problem:

$$\begin{aligned} \min_{T} \quad & \frac{(\Gamma+1)F(T)+\sigma}{T} \\ \text{s.t.} \quad & T \geq 1. \end{aligned} \tag{5.26}$$

Considering the original objective in (5.2), the problem (5.26) can be regarded as finding the optimal look-ahead window size such that an upper bound of the objective function in (5.2) is minimized (according to Proposition 5.5). The solution to (5.26) is the optimal window size to look-ahead so that (in the worst case) the cost is closest to the cost from the optimal configuration $\boldsymbol{\pi}^*$.

5.5.2 Characteristics of the Problem in (5.26)

In the following, we study the characteristics of (5.26). To help with the analysis, we interchangeably use variable T to represent either a discrete or a continuous variable. We define a continuous convex function $G(T)$, where $T \geq 1$ is a continuous variable. The function $G(T)$ is defined in such a way that $G(T) = F(T)$ for all the discrete values $T \in \{1, 2, ...\}$, i.e. $G(T)$ is a *continuous time extension* of $F(T)$. Such a definition is always possible by connecting the discrete points in $F(T)$. Note that we do not assume the continuity of the derivatives of $G(T)$, which means that $\frac{dG(T)}{dT}$ may be non-continuous and $\frac{d^2G(T)}{dT^2}$ may be $+\infty$. However, these do not affect our analysis below. We will work with continuous values of T in some parts and will discretize it when appropriate.

We define a function $\theta(T) \triangleq \frac{(\Gamma+1)G(T)+\sigma}{T}$ to represent the objective function in (5.26) after replacing $F(T)$ with $G(T)$, where T is regarded as a continuous variable. We take the logarithm of $\theta(T)$, yielding

$$\ln\theta = \ln\left((\Gamma+1)G(T)+\sigma\right) - \ln T. \tag{5.27}$$

Taking the derivative of $\ln\theta$, we have

$$\frac{d\ln\theta}{dT} = \frac{(\Gamma+1)\frac{dG(T)}{dT}}{(\Gamma+1)G(T)+\sigma} - \frac{1}{T}. \tag{5.28}$$

We set (5.28) equal to zero, and rearrange the equation, yielding

$$\Phi(T) \triangleq (\Gamma+1)T\frac{dG(T)}{dT} - (\Gamma+1)G(T) - \sigma = 0. \tag{5.29}$$

Proposition 5.6. *Let T_0 denote a solution to (5.29), if the solution exists, then the optimal look-ahead window size T^* for problem (5.26) is either $\lfloor T_0 \rfloor$ or $\lceil T_0 \rceil$, where $\lfloor x \rfloor$ and $\lceil x \rceil$ respectively denote the floor (rounding down to integer) and ceiling (rounding up to integer) of x .*

Proof. Taking the derivative of $\Phi(T)$, we get

$$\frac{d\Phi}{dT} = (\Gamma+1)T\frac{d^2G(T)}{dT^2} \geq 0 \tag{5.30}$$

where the last inequality is because $G(T)$ is convex. This implies that $\Phi(T)$ is non-decreasing with T. Hence, there is at most one consecutive interval of T (the interval may only contain one value) such that (5.29) is satisfied. We denote this interval by $[T_-, T_+]$, and a specific solution to (5.29) is $T_0 \in [T_-, T_+]$.

We note that $\frac{d\ln\theta}{dT}$ and $\Phi(T)$ have the same sign, because $\frac{d\ln\theta}{dT} \lesseqgtr 0$ yields $\Phi(T) \lesseqgtr 0$ and vice versa, which can be seen from (5.28) and (5.29). When $T < T_-$, we have $\Phi(T) < 0$ and hence $\frac{d\ln\theta}{dT} < 0$; when $T > T_+$, we have $\Phi(T) > 0$ and hence $\frac{d\ln\theta}{dT} > 0$. This implies that $\ln\theta$, thus $\theta(T)$, keeps decreasing with T until the optimal solution is reached, and afterwards it keeps increasing with T. It follows that the minimum value of $\theta(T)$ is attained at $T \in [T_-, T_+]$. Because $T_0 \in [T_-, T_+]$ and T^* is a discrete variable, we complete the proof. □

Note that we do not consider the convexity of $\theta(T)$ in the above analysis. From

the proof of Proposition 5.6, we can also conclude the following corollary.

Corollary 5.1. *For window sizes T and $T+1$, if $\theta(T) < \theta(T+1)$, then the optimal size $T^* \leq T$; if $\theta(T) > \theta(T+1)$, then $T^* \geq T+1$; if $\theta(T) = \theta(T+1)$, then $T^* = T$.*

5.5.3 Finding the Optimal Solution

According to Proposition 5.6, we can solve (5.29) to find the optimal look-ahead window size. When $G(T)$ (and $F(T)$) can be expressed in some specific analytical forms, the solution to (5.29) can be found analytically. For example, consider $G(T) = F(T) = \beta T^{\alpha}$, where $\beta > 0$ and $\alpha > 1$. In this case, $T_0 = \left(\frac{\sigma}{(\Gamma+1)\beta(\alpha-1)}\right)^{\frac{1}{\alpha}}$, and $T^* = \arg\min_{T\in\{\lfloor T_0\rfloor,\lceil T_0\rceil\}} \theta(T)$. One can also use such specific forms as an upper bound for a general function.

When $G(T)$ (and $F(T)$) have more general forms, we can perform a search on the optimal window size according to the properties discussed in Section 5.5.2. Because we do not know the convexity of $\theta(T)$ or $\Phi(T)$, standard numerical methods for solving (5.26) or (5.29) may not be efficient. However, from Corollary 5.1, we know that the local minimum point of $\theta(T)$ is the global minimum point, so we can develop algorithms that use this property.

Because the optimal window size T^* takes discrete values, we can perform a discrete search on $T \in \{1, 2, ..., T_m\}$, where $T_m > 1$ is a pre-specified upper limit on the search range. We then compare $\theta(T)$ with $\theta(T+1)$ and determine the optimal solution according to Corollary 5.1. One possible approach is to use binary search, as shown in Algorithm 5.4, which has time-complexity of $O(\log T_m)$.

Remark: We note that the exact value of Γ may be difficult to find in practice, and (5.20) is an upper bound which may have a gap from the actual value of the left hand-side of (5.20). Therefore, in practice, we can regard Γ as a tuning parameter, which can be tuned so that the resulting window size T^* yields good performance.

Algorithm 5.4 Binary search for finding optimal window size

```
Initialize variables T_- ← 1 and T_+ ← T_m
repeat
    T ← ⌊(T_- + T_+)/2⌋
    if θ(T) < θ(T+1) then
        T_+ ← T
    else if θ(T) > θ(T+1) then
        T_- ← T + 1
    else if θ(T) = θ(T+1) then
        return T //Optimum found
    end if
until T_- = T_+
return T_-
```

For a similar reason, the parameter σ can also be regarded as a tuning parameter in practice.

5.6 Simulation Results

In this section, we evaluate the performance of the proposed approach with simulations. We assume that there exist a backend cloud (with index n_0) and multiple MMCs. A service instance can be placed either on one of the MMCs or on the backend cloud. We first define

$$R(y) \triangleq \begin{cases} \frac{1}{1-\frac{y}{Y}}, & \text{if } y < Y \\ +\infty, & \text{if } y \geq Y \end{cases} \tag{5.31}$$

where Y denotes the capacity of a single MMC. Then, we define the local and migration costs as in (5.5), (5.6), with

$$u_{n,t}(y_n(t)) \triangleq \begin{cases} \tilde{g} y_n(t), & \text{if } n = n_0 \\ y_n(t) R(y_n(t)) + g r_n(t), & \text{if } n \neq n_0 \end{cases} \tag{5.32}$$

$$
\begin{aligned}
&w_{nh,t}(y_n(t-1), y_h(t), z_{nh}(t)) \\
&\triangleq \begin{cases} \tilde{h} z_{nh}(t), & \text{if } n = n_0 \text{ or/and } h = n_0 \\ z_{nh}(t)\left(R(y_n(t)) + R(y_h(t))\right) + h s_{nh}(t), & \text{else} \end{cases}
\end{aligned} \tag{5.33}
$$

where $y_n(t)$ and $z_{nh}(t)$ are defined in the same way as in Section 5.4.2.1, $r_n(t)$ is the sum of the communication distances between each instance running on cloud n all users connected to this instance, $s_{nh}(t)$ is the communication distance between clouds n and h multiplied by the number migrated instances from cloud n to cloud h, and $\tilde{g}, g, \tilde{h}, h$ are simulation parameters. The communication distance is expressed in the number of hops on the communication network.

In the cost functions defined in (5.32) and (5.33), as in Section 5.4.2.7 (but with migration cost here), the local and migration costs involving the backend cloud n_0 are linear in the corresponding resource consumptions $y_n(t)$ and $z_{nh}(t)$. When not involving the backend cloud, the cost functions have two terms. The first term containing $R(\cdot)$ can be explained as related to the queuing delay of data processing/transmission, where we note that the function $R(\cdot)$ has a similar form as the average queueing delay expression from queueing theory, and the additional coefficient $y_n(t)$ or $z_{nh}(t)$ scales the delay by the total amount of workload so that the experience of all service instances (hosted at the cloud or being migrated) are considered. This expression is also a widely used objective (such as in [17]) which pushes the system towards a load-balanced state. The second term is related to the distance of data transmission or migration.

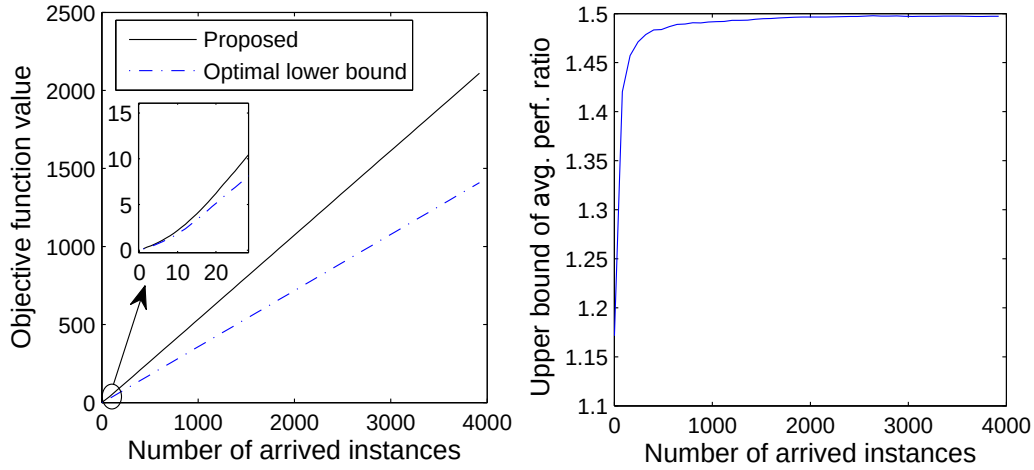
Note that the above defined cost functions are heterogeneous, because the cost definitions are different depending on whether the backend cloud is involved or not. Therefore, we cannot directly apply the existing MDP-based approaches to solve this problem.

5.6.1 Synthetic Arrivals and Departures

To evaluate how much worse the online placement (presented in Section 5.4) performs compared to the optimal offline placement (presented in Section 5.3), we first consider a setting with synthetic instance arrivals and departures. For simplicity, we ignore the migration cost and set $g = 0$ to make the local cost independent of the distance $r_n(t)$. We set $Y = 5$, $\tilde{g} = 3$, and the total number of clouds $N = 5$ among which one is the backend cloud. We simulate 4000 arrivals, where the local resource consumption of each arrival is uniformly distributed within interval $[0.5, 1.5]$. Before a new instance arrives, we generate a random variable ζ that is uniformly distributed within $[0, 1]$. If $\zeta < 0.1$, one randomly selected instance that is currently running in the system (if any) departs. We only focus on the cost in a single timeslot and assume that arrival and departure events happen within this slot. The online placement greedily places each instance, while the offline placement considers all instances as an entirety. We compare the cost of the proposed online placement algorithm with a lower bound on the cost of the optimal placement. The optimal lower bound is obtained by solving an optimization problem that allows every instance to be arbitrarily split across multiple clouds, in which case the problem becomes a convex optimization problem due to the relaxation of integer constraints.

The simulation is run with 100 different random seeds. Fig. 5.4 shows the overall results. We see that the cost is convex increasing when the number of arrived instances is small, and it increases linearly when the number of instances becomes large, because in the latter case, the MMCs are close to being overloaded and most instances are placed at the backend cloud. In Fig. 5.4(b), we show the average performance ratio, defined as

$$\Gamma_{\text{avg}} \triangleq \frac{\text{mean}\left(\widetilde{D}(\mathbf{x})\right)}{\text{mean}\left(\widetilde{D}(\mathbf{x}^*)\right)} \tag{5.34}$$



(a) (b)

Figure 5.4: Results with synthetic traces: (a) objective function value, (b) average performance ratio.

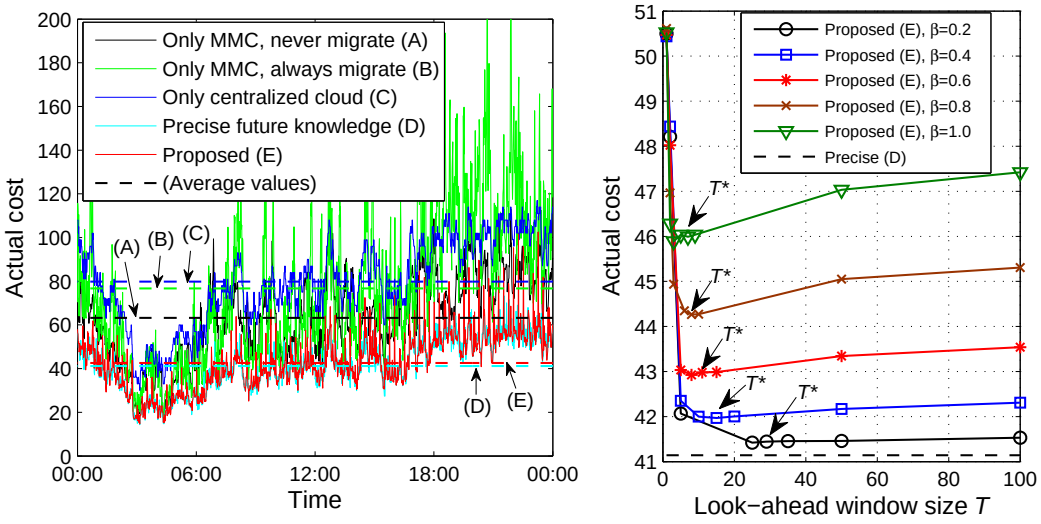
which is an analogy for the competitive ratio. The difference is that the average performance ratio quantifies the performance gap in an average sense, while competitive ratio quantifies the worst-case performance gap. Because we cannot find the real worst case from a limited number of simulation instances, we plot the average performance here. We see that the average performance ratio converges with increasing number of instances, which supports our analysis in Section 5.4.2.7.

5.6.2 Real-World Traces

To further evaluate the performance while considering the impact of prediction errors and look-ahead window size, we perform simulations using real-world San Francisco taxi traces obtained on May 31, 2008 [45, 46]. We assume that the MMCs are deployed according to a hexagonal cellular structure, and the distance between the center points of adjacent cells is 1000 m. We consider $N - 1 = 91$ cells (thus MMCs), one backend cloud, and 50 users (taxis) in total and not all the users are active (which can be derived from the dataset) at a given time. A user may require at most one service at a time from the cloud when it is active, where the duration that

each active user requires (or, does not require) service is exponentially distributed with a mean value of 50 slots (or, 10 slots). When a user requires service, we assume that there is a service instance for this particular request (independent from other users) running on one of the clouds. The local and migration (if migration occurs) resource consumptions of each such instance are set to 1. We assume that the online algorithm has no knowledge on the departure time of instances and set $T_{\text{life}} = \infty$ for all instances. Note that the taxi locations in the dataset are unevenly distributed, with most taxis in the central area of San Francisco, so it is still possible that one MMC hosts multiple services although the maximum possible number of instances (50) is smaller than the number of MMCs (91). The distance metric (for evaluating $r_n(t)$ and $s_{nh}(t)$) is defined as the minimum number of hops between two locations on the cellular structure. The physical time corresponding to each slot is set to 60 s. We set the parameters $\Gamma = 1.5, \sigma = 2, Y = 5, \tilde{g} = \tilde{h} = 3, g = h = 0.2$. The cost prediction error is assumed to have an upper bound in the form of $F(T) = \beta T^{\alpha}$ (see Section 5.5.3), where we fix $\alpha = 1.1$. The prediction error is generated randomly while ensuring that the upper bound is satisfied. The simulation results are shown in Fig. 5.5.

In Fig. 5.5(a), we can see that the result of the proposed online placement approach (E) performs close to the case of online placement with precise future knowledge (D), where approach D assumes that all the future costs as well as instance arrival and departure times are precisely known, but we still use the online algorithm to determine the placement (i.e., we greedily place each instance), because the offline algorithm is too time consuming due to its high complexity. The proposed method E also outperforms alternative methods including only placing on MMCs and never migrate the service instance after initialization (A), always following the user when the user moves to a different cell (B), as well as always placing the service instance on the backend cloud (C). In approaches A and B, the instance placement is



(a) (b)

Figure 5.5: Results with real-world traces (where the costs are summed over all clouds, i.e., the $A(t)$ values): (a) Actual costs at different time of a day, where $\beta = 0.4$ for the proposed method E. The arrows point to the average values over the whole day of the corresponding policy. (b) Actual costs averaged over the whole day.

determined greedily so that the distance between the instance and its corresponding user is the shortest, subject to the MMC capacity constraint Y so that the costs are finite (see (5.31)). The fluctuation of the cost during the day is because of different number of users that require the service (thus different system load).

In Fig. 5.5(b), we show the average cost over the day with different look-ahead window sizes and β values, where the average results from 8 different random seeds are shown. Recall that β quantifies the prediction error upper bound, and a large β indicates a large prediction error. The main observation from Fig. 5.5(b) is that for each value of β we consider in the simulation, the optimal window size (T^*, pointed by arrows in the figure) found from the method proposed in Section 5.5 is close to the actual optimal window size that brings the lowest cost (which can be seen in the cost plot for the particular value of β). This implies that the proposed method for finding T^* is reasonably accurate. We also see that different β values yield different optimal window sizes; the optimal window size is small when β is large. This is because a

larger β corresponds to a larger prediction error. When the prediction error is large, the algorithm should not look too far ahead into the future, to avoid bringing in a large error which would bias the optimal placement decision in both short and long term.

5.7 Summary

In this chapter, we have studied the dynamic service placement problem utilizing predicted future costs. Noting that the actual future costs cannot be obtained precisely, we have proposed a method that utilizes predicted future costs with a known upper bound on the prediction error to make placement decisions that are close to the true optimum. The method includes finding the optimal window size to look ahead as well as finding the optimal configuration within each look-ahead window. For the latter aspect, we have considered both the optimal offline placement and an approximate online placement with provable performance guarantee.

CHAPTER 6

Emulation-Based Study

6.1 Introduction

Summing up all the previous chapters, we have proposed algorithms for both initial service placement and real-time service migration (see Section 1.1 for an overview on these two subproblems). The analysis started from a theoretical perspective, with practical considerations discussed subsequently in some chapters. The performances of algorithms have also been evaluated using real-world user mobility traces in Chapters 4 and 5. These contributions and discussions are from both theoretical and practical perspectives, which justify the reasoning of algorithm design, performances of proposed algorithms, and practical applicability of the algorithms.

In this chapter, we aim to further push the theoretical results one step closer to practice. We present a framework for conducting experiments in an emulated cloud environment that contains one centralized core cloud and multiple MMCs, which jointly cover an area that contains multiple mobile users. The emulation is performed using the Common Open Research Emulator (CORE) [57, 58] which embeds the Extendable Mobile Ad-hoc Network Emulator (EMANE) [59]. In CORE/EMANE, each network (physical) node is encapsulated into a virtual container which can run a piece of code independent of other nodes. These network (physical) nodes are then connected via emulated network (physical) links for which the quality may vary over time and may be related to the location of nodes (for wireless links). Implementing algorithms in CORE enables more realistic experimentation compared to simulation,

thus providing insight into what behavior or performance we can expected when implementing these algorithms in real systems.

The emulation-based study we consider in this chapter has the following key differences from the theoretical work in previous chapters:

1. In the emulation, the parameters related to the transmission and migration costs are estimated based on real-time network measurements, which can be subject to fluctuations and inaccuracies.

2. A fully distributed system is considered in the emulation. No node (i.e., core cloud, MMC, or user) in the network has full control over other nodes. They can only communicate with each other by exchanging control messages over the communication network.

3. The emulation considers realistic communication links that may experience loss and delay as a result of congestion and/or physical layer effects. Control messages may also be delayed or lost.

The ultimate goal of emulation study is to evaluate the theoretical findings (such as those presented in previous chapters) in realistic settings where some assumptions made for theoretical tractability may not hold. However, non-trivial effort is needed to achieve this ultimate goal, because we need to properly translate the theoretical setting into a practical setting. For example, we have to collect all the information an algorithm needs for decision making, which by itself requires notable implementation effort because the transmission cost, for example, can be related to the users' associated basestations, signal strength of wireless channels, topology of backhaul network, etc. In a practical emulation environment such as CORE/EMANE, developing a program for obtaining all such information is not straightforward but doable if sufficient time is allowed. For simplicity, we propose an initial emulation framework in this chapter, and perform emulation with some simple service migration

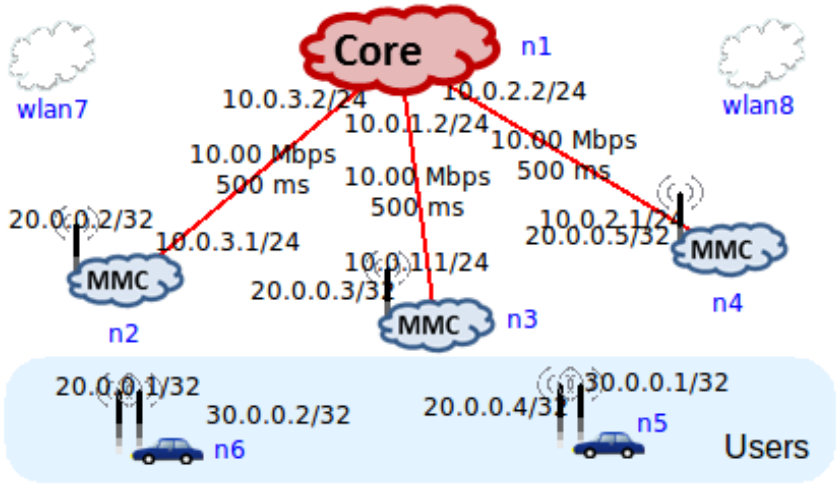


Figure 6.1: System architecture for CORE emulation.

algorithms. This sets the foundation of emulating more sophisticated algorithms with realistic applications in the future.

6.2 System Architecture

In the CORE emulation, we abstract the application scenario depicted in Fig. 1.1 into the system architecture shown in Fig. 6.1. In Fig. 6.1, node `n1` is the core centralized cloud; nodes `n2`, `n3`, and `n4` are MMCs; and nodes `n5` and `n6` are users. Note that this is only an example and the number of MMCs and users can be arbitrary in the emulation.

6.2.1 Network Connection and User Mobility

The core cloud is connected with each MMC via a persistent link configured with a relative low bandwidth and a relatively large delay.

Each MMC and each user has a wireless interface, which has an IP address of `20.0.0.x` and is configured by the configuration node `wlan7` in Fig. 6.1. Throughout the emulation, we use a fixed range radio propagation model. A pair of nodes within the specified range can communicate with each other, and they can-

not communicate if they are outside the specified range, where a node can be either a user or an MMC. More realistic propagation models can be considered in the future using functionalities of EMANE. The OSPF-MDR routing protocol [60] is used for routing among the wireless nodes. A pair of nodes may communicate either in single-hop or multi-hop depending on their distance.

We consider in the emulation the case where MMCs are static and users are mobile. This is only to ease the emulation setup, and the same design principle can be applied to scenarios where MMCs are also mobile. We employ the EMANE event service to use existing traces to govern the mobility of users. The mobility is described by an Emulation Event Log (EEL) file. To receive messages from EMANE, each user has an additional wireless interface with an IP address of `30.0.0.x`, which is configured by the configuration node `wlan8`. This interface is used only for communicating with EMANE (which is executed outside the emulator in the emulator's host machine) so that the user locations remain updated in real time according to the trace file. It does not participate in any actual communication in the emulated network.

With the above setting, the core cloud, MMCs, and users can communicate with each other. The wireless connections are also updated in real-time based on locations of users. Users do not have a direct connection with the core cloud, but they can connect to the core via an MMC.

6.2.2 Service Model

We consider a relatively simple service model as presented next, while noting that more sophisticated cases can be built based on the emulation framework we present here.

We assume that each user is running one (independent) service instance in the cloud, and the terms "service" and "service instance" are exchangeably used. Ser-

vice components can only be placed on MMCs and no service component is placed on the core cloud. This simplifies the control procedure of service placement, while more sophisticated cases involving the core cloud can be considered in the future by extending the current framework. For now, the role of the core cloud is to act as a controller for service placement, which is a relatively robust setup for a centralized control approach because the system can still work even if some MMCs fail to function.

We consider a delay-sensitive situational awareness application, such as the face recognition application introduced in Section 1.1. In such applications, each user regularly sends data (e.g., images as in the face recognition application) describing its current surrounding to the MMC that is hosting its service. The MMC analyzes user data[1] to extract user situation (e.g., detecting and tagging objects and faces in images) and sends the results back to the user, also in a regular manner. We abstract this process as UDP packet transfer between the user and the MMC, and we name these packets as *service packets* to distinguish them from control packets introduced later. The service packet transmission is initiated by the MMC. The MMC sends a `MMC_SERVICE_PACKET` to each user (for which it is hosting a service) at a pre-specified interval. After receiving this packet, the user responds with its own `USER_SERVICE_PACKET` that contains newly measured data to the MMC.

The performance metric of the service is the round-trip delay of transmitting service packets from the MMC to the user and then back to the MMC. This mimics situational awareness applications that need to collect user data as quickly as possible so that rapid decisions can be made. This round-trip delay can be seen as the transmission cost between the user and the MMC.

[1]Note that this is a simplified version of the face recognition example presented in Section 1.1. We assume here that the face recognition module and the database are also placed on the MMC, which can be a feasible configuration when the database is not too large. This is a simplification and more general cases can be considered in the future.

For simplicity, in the emulation framework proposed in this chapter, we assume that each MMC can host any number of services. In more realistic scenarios, different services may consume different amount of resources of the hosting MMC, and the number of services that a MMC can host will be subject to its capacity limit. We envision that algorithms presented in previous chapters can be used for making placement decisions in such capacity-limiting scenarios. Most of these algorithms either explicitly or implicitly incorporate capacity constraints, and are able to avoid placing new services on MMCs that result in violation of the capacity constraints.

We also assume that service migration can be completed instantaneously, and leave the more realistic case where there is a cost (such as interruption of service, bandwidth consumption for transferring application data) associated with migration as future work. As an indicator of the migration cost, we record the number of migrations in the emulation results in Section 6.4. We also note that for real applications, the migration cost can be related to the state of the application at the time when migration is performed. While the algorithms presented in previous chapters have considered the migration cost, they have not considered the application state which could impact the migration cost. This aspect is worth studying in the future. The emulation framework we present in this chapter can be an essential tool for such a study, because we can run real applications on our emulated MMC platform, which would be very helpful for studying the interplay between application state and migration cost.

6.3 Packet Exchange and Placement Control

6.3.1 Control Messages

Control messages need to be exchanged for performance measurement and controlling the placement of services. All messages are sent in UDP packets. The different

types of messages are described as follows.

6.3.1.1 Beacon Messages from Core Cloud

At a pre-specified interval, the centralized core cloud sends out a beacon message `CORE_BEACON_MMC` to each MMC, and it also sends out a beacon message `CORE_BEACON_USER` to each user. This is to notify MMCs and users of some status information, including the set of users each MMC should currently serve, the set of users each MMC should probe for delay measurement, etc. Before sending these beacon messages, the core cloud makes decisions such as where to place the service for each user.

6.3.1.2 Connection Request

In order for the core cloud to know which MMCs and users are currently present in the system, each MMC sends `MMC_CONNECT` and each user sends `USER_CONNECT` at a pre-specified interval.

6.3.1.3 Delay Probing

Probing for delay measurement is initiated by each MMC at a pre-specified interval. Every MMC probes a set of users as instructed by the core cloud. This set of users should be within a specific proximity of the MMC, so that the MMC is potentially suitable of running services for these users. An MMC first sends `MMC_DELAY_PROBE` to all the users it intends to probe. Upon receiving this message, each user immediately replies with a `USER_DELAY_PROBE` to the MMC that sent this message. The MMC calculates the round-trip delay for the particular user and sends the result to the core cloud with an `MMC_USER_DELAY_MEASURED` message. After the core cloud has received this delay information, it stores it in a list so that it can be used for making service placement decisions before sending beacon

Algorithm 6.1 Procedure at the core cloud

```
1: loop
2:    if timer t(CORE_BEACON_MMC) expired then
3:       Update beacon information, such as the placement of services for all users
         (obtained based on the recorded delay measurements)
4:       Send CORE_BEACON_MMC to each MMC
5:       Reset timer t(CORE_BEACON_MMC)
6:    end if
7:    if timer t(CORE_BEACON_USER) expired then
8:       Update beacon information
9:       Send CORE_BEACON_USER to each user
10:      Reset timer t(CORE_BEACON_USER)
11:   end if
12:   if received MMC_USER_DELAY_MEASURED then
13:      Update the recorded delay statistics
14:   end if
15:   if received MMC_CONNECT or USER_CONNECT then
16:      Update the MMC or user record
17:   end if
18: end loop
```

messages.

We note that one may think of using service packets for delay measurement. However, service packets are only transmitted between a user and the particular MMC that is hosting its service, and the delay of transmitting packets between the user and other MMCs cannot be obtained from service packets. Therefore, we introduce additional packets for delay probing, which is usually substantially shorter than service packets.

6.3.2 Packet Exchange and Control Procedure

The detailed procedures that are executed at the core cloud, MMC, and user are respectively shown in Algorithms 6.1, 6.2, and 6.3, where we use the notation $t(\texttt{A})$ to denote the timer for packet A, the packet A is sent when $t(\texttt{A})$ expires.

Algorithm 6.2 Procedure at each MMC

```
1: loop
2:    if timer t(MMC_CONNECT) expired then
3:       Send MMC_CONNECT to the core cloud
4:       Reset timer t(MMC_CONNECT)
5:    end if
6:    if timer t(MMC_DELAY_PROBE) expired then
7:       Send MMC_DELAY_PROBE to each user it intends to probe
8:       Reset timer t(MMC_DELAY_PROBE)
9:    end if
10:   if timer t(MMC_SERVICE_PACKET) expired then
11:      Send MMC_SERVICE_PACKET to each user it is serving
12:      Reset timer t(MMC_SERVICE_PACKET)
13:   end if
14:   if received USER_DELAY_PROBE then
15:      Calculate the round-trip delay and send the result via
         MMC_USER_DELAY_MEASURED to the core cloud
16:   end if
17: end loop
```

Algorithm 6.3 Procedure at each user

```
1: loop
2:    if timer t(USER_CONNECT) expired then
3:       Send USER_CONNECT to the core cloud
4:       Reset timer t(USER_CONNECT)
5:    end if
6:    if received t(MMC_DELAY_PROBE) then
7:       Send USER_DELAY_PROBE to the originating MMC
8:    end if
9:    if received t(MMC_SERVICE_PACKET) then
10:      Send USER_SERVICE_PACKET to the originating MMC
11:   end if
12: end loop
```

6.3.3 Service Placement Decisions

In the emulation, we consider three different policies for deciding the service placement. These are described as follows.

6.3.3.1 Always Migrate (AM)

In the AM policy, the core cloud always looks at its most recently received delay measurements, and places the service for each user to the MMC that has the lowest delay as measured by the latest probe. This policy puts strong emphasis on minimizing the transmission cost (round-trip delay), and does not consider the migration cost (number of migrations).

6.3.3.2 Infrequently Migrate (IM)

The IM policy is similar to the AM policy, except that migration can only occur at an interval denoted by τ. The value of τ is normally large compared to the delay probing interval, so that migration happens infrequently. The initial placement of services is not restricted by τ, which means that if a user is not served by any cloud previously, its service can be placed immediately after the core cloud (controller) recognizes that it has established connection with an MMC. The IM policy attempts to bound the migration cost, while the transmission cost may be large because services may not be migrated soon enough after the transmission delay has changed. The IM policy mimics the never migrate policy used for comparison in previous chapters. We use IM instead of never migrate here to avoid the system being trapped in the initial state.

6.3.3.3 Moving Average + Hysteresis (MAH)

The MAH policy does not use the instantaneous delay measurement for service placement and migration decisions. Instead, it uses an exponentially-weighted moving averaged delay for decision making. For each MMC i and user j, the moving average is performed according to

$$D_{ij}(m) = \alpha D_{ij}(m-1) + (1-\alpha) d_{ij}(m) \tag{6.1}$$

where $d_{ij}(m)$ is the mth round-trip delay for MMC-user pair (i, j) that has been received by the core cloud, $D_{ij}(m)$ is the delay used for decision making after receiving the mth and before receiving the $(m + 1)$th measurement, $0 \leq \alpha \leq 1$ is a controllable parameter for moving average computation.

The MAH policy also has a hysteresis delay value $\varepsilon \geq 0$. If the service for user j is previously running at MMC i, a migration to MMC i' only occurs when $D_{i'j}(m) < D_{ij}(m) - \varepsilon$. This prevents frequent migration between different MMCs.

We can see that the MAH policy can be configured to achieve various trade-offs between the transmission and migration costs by tuning the values of α and ε. It is used here as an alternative representative of the theoretical results in previous chapters, which jointly consider the transmission and migration costs. The MAH policy is simpler than the algorithms in previous chapters, but also less robust due to the existence of parameters α and ε that are subject to tuning. We also note that the AM policy is a special case of the MAH policy under $\alpha = \varepsilon = 0$.

6.4 Emulation Scenario and Results

We created an emulation scenario that consists of 20 users and 16 MMCs distributed in the San Francisco area, as shown in Fig. 6.2. We assume that the MMCs are static. The user mobility is generated using the San Francisco taxi dataset [45, 46] that has also been used in earlier chapters. To reduce the required emulation time and see the impact of user mobility, we compressed the timescale by a factor of 6, so that 6 seconds in the original trace is one second in the emulation. We note that the San Francisco taxi traces is only one representative of a real-world mobility pattern, which is the mobility of vehicles moving in a city area. Some other mobility patterns that can be helpful for evaluating the performance of MMCs include vehicles moving on highways, pedestrians moving in cities, etc. It can be worthwhile to carry out an

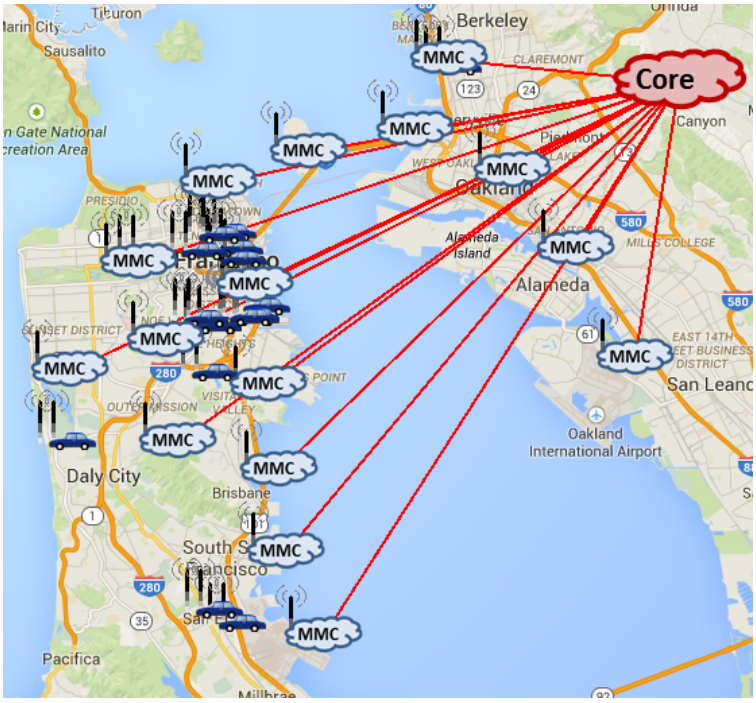


Figure 6.2: Emulation scenario (source of map: https://maps.google.com/).

in-depth study using different mobility patterns in the future.

We ran the emulation for $15,000$ seconds, in which we applied either the AM, IM, or MAH policy for service placement. The interval of sending `CORE_BEACON_MMC`, `CORE_BEACON_USER`, and `MMC_DELAY_PROBE` is specified by parameter T which take different values in the emulation. The value of T specifies the interval of delay probing and service placement update. Other parameter settings in the emulation are summarized in Table 6.1.

We first set $T = 2\,\text{s}$ and focus on the instantaneous results in the first $3,000$ seconds of emulation. The instantaneous round-trip delay of service packets under different policies is shown in Fig. 6.3, where the results for service packets corresponding to different MMC-user pairs are merged into the same timeline and plotted in one graph. We can see that there are many spikes in the figure, which indicates that the delay exhibits large variation. Reasons for this include variation in network traffic, diversity of user and service locations, as well as the change of user and service locations over time. The last reason is obvious from Fig. 6.3(b), where the delay has

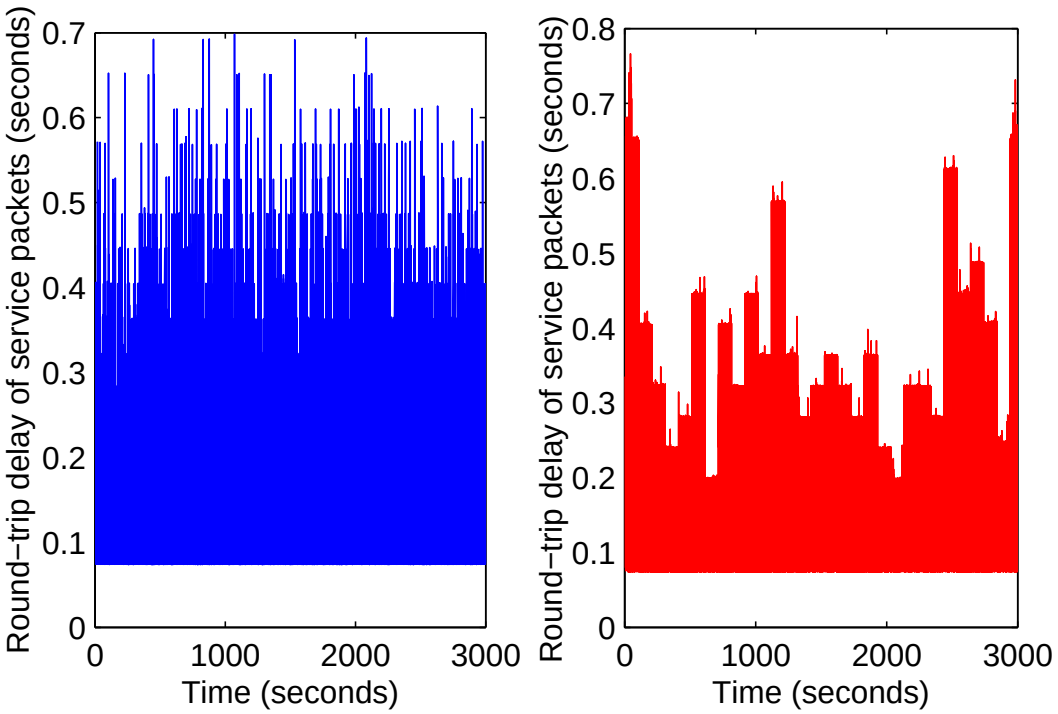
Table 6.1: Emulation setup

Parameter name	Value
Wireless communication bandwidth	10 Mbps
One-hop wireless communication range	6,000 m
One-hop wireless communication delay	20 ms
Bandwidth of link connecting MMC and core cloud	10 Mbps
Delay of link connecting MMC and core cloud	500 ms
Size of `MMC_SERVICE_PACKET`	1,000 Bytes
Size of `USER_SERVICE_PACKET`	50,000 Bytes
Interval of sending `MMC_SERVICE_PACKET`	2 s
Interval of sending `MMC_CONNECT`	10 s
Interval of sending `CORE_BEACON_MMC`, `CORE_BEACON_USER`, and `MMC_DELAY_PROBE`	T (variable)
Parameter τ in IM policy	100 s
Parameter α in MAH policy	0.5
Parameter ε in MAH policy	10 ms

a block pattern because the IM policy only migrates (and thereby changing service locations) once in 100 seconds.

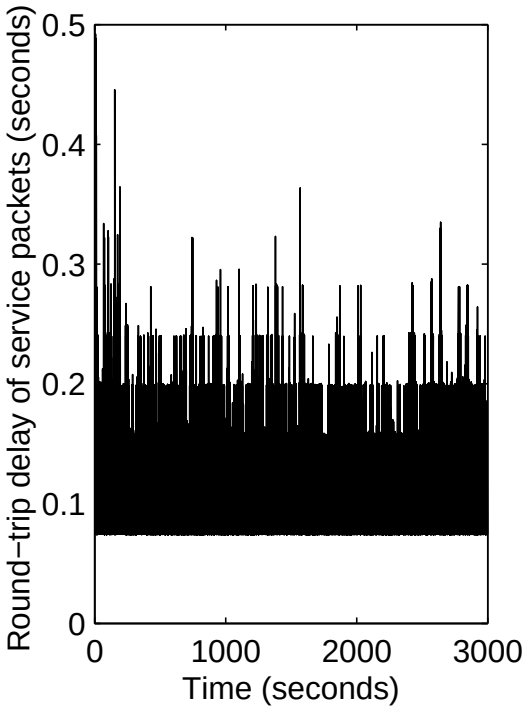
To obtain a more comprehensible set of results, we perform cumulative moving average with a window size of 50 s on the instantaneously measured data. At each time instant (in the resolution of one second), we look back to the past 50 s (or up to the emulation start time, whichever is later) and plot the average result within this window. The moving averaged delay is shown in Fig. 6.4(a), and Fig. 6.4(b) shows the average number of service migrations within the 50 s window size. We can see that compared to the AM and MAH policies, the IM policy may cause blocks of large delays, because services may be placed at non-optimal locations for a long time.

The overall performance with different values of T is shown in Fig. 6.5, where the results are collected starting from 1,000 s emulation time to remove the impact of variation in the initialization time under different T. We see that as expected, the AM policy always has the largest number of migrations. It is interesting that round-trip delay of the AM policy is also generally larger than that of the MAH policy, mainly because the AM policy uses instantaneous delay measurements for



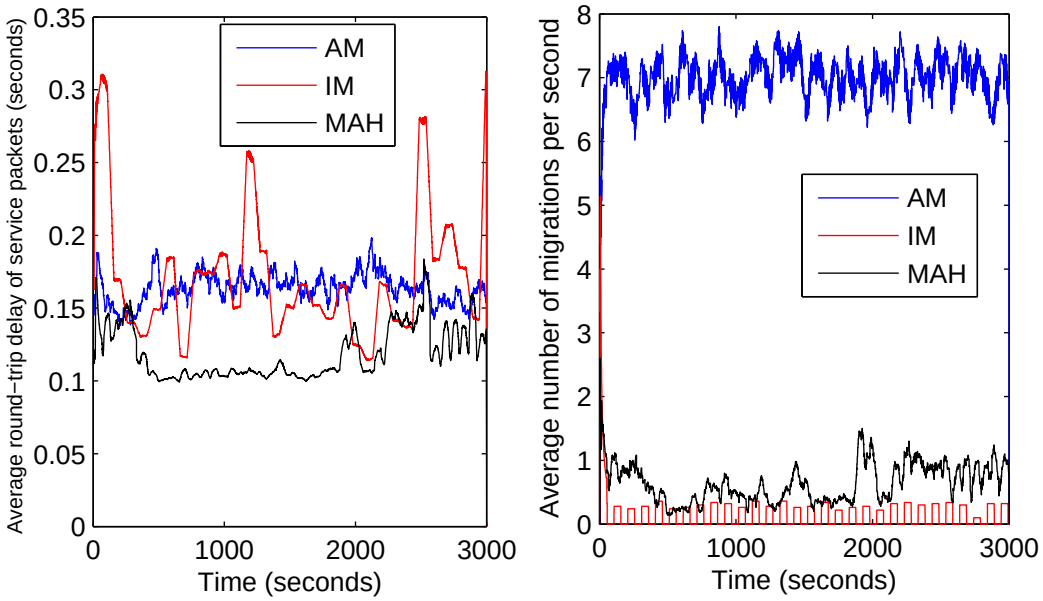
(a)

(b)



(c)

Figure 6.3: Instantaneous round-trip delays of service packets for the first $3,000$ s of emulation with $T = 2$ s: (a) AM policy, (b) IM policy, (c) MAH policy.

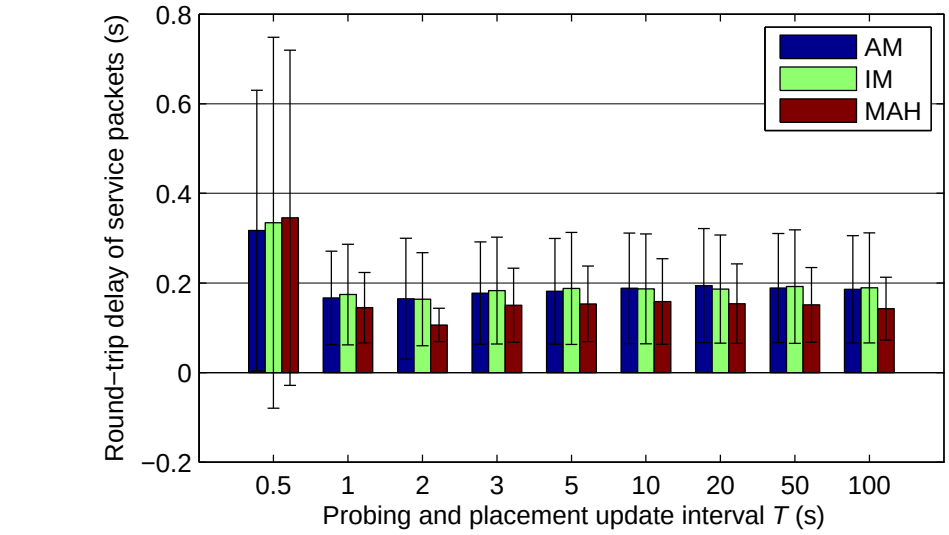


(a) (b)

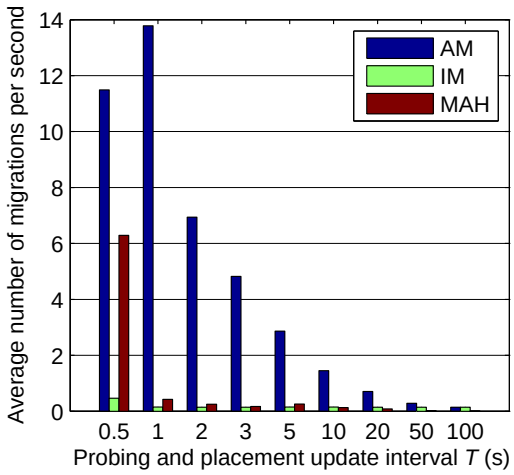
Figure 6.4: Moving average results for the first $3,000$ s of emulation with $T = 2$ s: (a) round-trip delay of service packets, (b) number of migrations.

service placement decisions, which may fluctuate substantially over time and cause the placement decision deviate away from the optimum. Instead, the MAH policy is more stable because it has the moving average and hysteresis building blocks. The delay performance of the MAH policy is also generally better than the IM policy, because services are migrated to good locations more frequently.

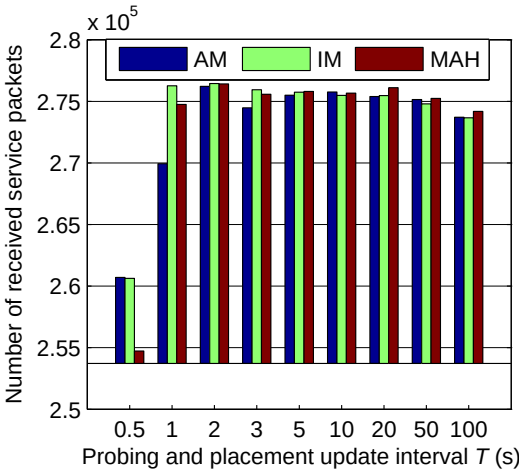
The performance is related to the value of the interval T. It is clear that $T = 0.5$ s brings the worst performance, in which case the control messages overload the network. For the MAH policy, its lowest delay and highest number of received service packets (which represents the packet success rate) is attained at $T = 2$ s, while giving an intermediate number of migrations. We also note that the average delay of the MAH policy is the lowest among all the policies when $T = 2$ s, also with lower standard deviation than all other cases. This implies that the MAH policy can be beneficial for delay-sensitive applications. When T is large, the delay performance does not vary significantly with different T, but the number of received service pack-



(a)



(b)



(c)

Figure 6.5: Overall results: (a) average round-trip delay of service packets (error bars denote the standard deviation), (b) average number of migrations per second, (c) total number of received service packets.

ets decreases with T. The network in this case is not overloaded, thus the delay is not very large. However, due to obsolete delay measurement and prolonged service placement update, the service location may be far away from user location, so that the connection between the MMC and the user has multiple hops, causing higher packet loss. The slight fluctuation in the performance with different values of T is due to randomness in the emulation.

6.5 Summary

In this chapter, we have taken an initial step to an emulation-based study of service placement/migration in an MMC environment containing mobile users. We have proposed a simple emulation framework that can be used as a foundation for more sophisticated emulations in the future.

CHAPTER 7

Conclusions and Future Work

7.1 Contributions and Conclusions

This thesis has focused on service placement in a networked cloud environment containing mobile micro-clouds (MMCs). The hierarchical structure of MMCs allows us to exploit and develop new algorithms for service graph placement, which are more efficient than existing approaches. Meanwhile, MMCs bring in much more user and infrastructure dynamics compared to a traditional cloud environment, and we have therefore proposed mechanisms for controlling the service placement/migration to cope with these dynamics. The main contributions of this thesis are summarized as follows.

7.1.1 Application Graph Placement

In Chapter 3, we have studied the placement of an incoming stream of application graphs onto a physical graph. With the goal of minimizing the maximum resource utilization at physical nodes and links, we have proposed an exact optimal algorithm for placing one linear application graph onto a tree physical graph, as well as approximation algorithms for placing tree application graphs onto tree physical graphs. When the maximum number of unplaced junction nodes on any path from the root to a leaf (in the application graph) is a constant, the proposed algorithm has polynomial time-complexity and provides polynomial-logarithmic (poly-log) worst-case performance bound. Besides the theoretical evaluation of the worst-case performance, we

have also shown the average performance via simulation. A combination of these results implies that the proposed method performs reasonably well on average and it is also robust in extreme cases. Moreover, the method we propose is relatively easy to implement in practice.

The results in Chapter 3 can be regarded as an initial step towards a more comprehensive study in this direction. Many constraints in the problem formulation are for simplicity of presentation, and can be readily relaxed for a more general problem. For example, the tree application and physical graph restriction is not absolutely essential for the applicability of the key concepts in our proposed algorithms. Modified algorithms based on similar ideas would work for more general graphs as long as the cycle-free constraint is satisfied. While we have not considered services leaving at some time after their arrival, our algorithm can be extended to incorporate such cases using the idea in [61]. The algorithm for cases with unplaced junction nodes is essentially considering the scenario where there exists some low-level placement (for each of the branches) followed by some high level placement (for the junction nodes). Such ideas may also be useful in developing practical distributed algorithms with provable performance guarantees.

7.1.2 MDP-Based Approach to Dynamic Service Migration

In Chapter 4, we have studied service migration in MMCs to cope with user mobility and network changes. The problem is formulated as an MDP, but its state space can be arbitrarily large. To make the problem tractable, we have considered reasonable simplifications for 1-D and 2-D user mobility models.

For the 1-D case, we have considered a constant cost model and proposed a threshold policy-based mechanism for service migration in MMCs. We have shown the existence of a threshold policy that is an optimal policy, and have proposed an algorithm for finding the optimal thresholds. The proposed algorithm has polynomial

time-complexity, which is independent of the discount factor γ. This is promising because the time-complexity of standard algorithms for solving MDPs, such as the value or policy iteration methods, is generally dependent on the discount factor, and those solutions can only be shown to have polynomial time-complexity when the discount factor is regarded as a constant [42].

For the 2-D case, we have considered a constant-plus-exponential cost model. We have reduced the general problem into an MDP that only considers a meaningful parameter, namely the distance between the user and service locations. The distance-based MDP has several structural properties that allow us to develop an efficient algorithm for finding its optimal policy. We have then shown that the distance-based MDP is a good approximation to scenarios where the users move in a 2-D space, which is confirmed by analytical and numerical evaluations. Afterwards, we have illustrated the application of our solution in practical scenarios where many theoretical assumptions are relaxed. Our evaluation based on real-world mobility traces of San Francisco taxis shows the superior performance of the proposed solution compared to baseline solutions.

The results of Chapter 4 provide an efficient solution to service migration in MMCs. Further, we envision that the approaches used in this chapter can be extended to a range of other problems that share similar properties. The highlights of our approaches include: a closed-form solution to the discounted sum cost of a particular class of MDPs, which can be used to simplify the procedure for finding the optimal policy; a method to approximate an MDP (in a particular class) with one that has much smaller state space, where the approximation error can be quantified analytically; and a method to collect statistics from the real-world environment to serve as parameters of the MDP.

7.1.3 Dynamic Service Placement with Predicted Future Costs

Noting that the MDP-based approach can be difficult to apply to cases with more general user mobility and cost functions, we have assumed the ability of predicting future costs to some known accuracy in Chapter 5. This allows us to develop a mechanism for dynamic service placement in more general scenarios, including those where service instances arrive and depart without prior knowledge. We have defined a window to specify the time to look (or predict) ahead. Cost prediction is performed at the beginning of each look-ahead window. Based on the predicted cost, we find the service placement sequence, where we have considered both the optimal offline placement and approximate online placement. The online placement algorithm has polynomial time-complexity and we have analytically shown its optimality gap. Its competitive ratio is a constant for certain types of cost functions. We have then studied how to find the optimal look-ahead window size. The performance of the proposed approach is verified via simulations using both synthetic mobility traces and real-world mobility traces of San Francisco taxis.

The theoretical framework used for analyzing the performance of online placement can be extended to incorporate more general cases, such as those where there exist multiple types of resources in each cloud. We envision that the performance results will be qualitatively similar. We also note that our framework can be applied to analyzing a large class of online resource allocation problems that have convex objective functions.

7.1.4 Emulation-Based Study

With the goal of studying the performance of the above theoretical results in a more practical setting, we have proposed an emulation framework based on CORE/EMANE in Chapter 6. Different from its previous chapters, Chapter 6

considers realistic exchange of control and service packets. The emulation results show several insightful observations, such as the impact of randomness and delay on the service placement performance in a realistic network setting.

7.2 Future Work

Some possible areas of future work are summarized as follows.

Queuing Aware Scheduling of Service Requests: The approaches presented in this thesis have focused on scenarios where users are continuously connected to services. In reality, user requests may be buffered at the cloud for some time before they are served. Service placement with consideration of queuing/buffering can be studied in the future. We envision that queuing-aware scheduling techniques, such as Lyapunov optimization [62], can be used for such study. One such attempt is recently reported in [63].

Distributed Decision Making: In the approaches presented in this thesis, decisions on service placement are made by a centralized controller. This is usually feasible in practice because we can always see the controller as a service running at one or multiple clouds. However, it is desirable to have a distributed control mechanism for the sake of robustness. Issues regarding distributed service placement can be studied in the future, where randomization techniques such as in [64] and [65] can be used.

Practical Aspects: As MMCs have not yet been largely deployed in practice, it is definitely worth studying its implementation issues and identifying new challenges from a practical angle, which in turn can introduce new problems that are worthwhile for theoretical study. The emulation framework proposed in Chapter 6 can be useful

for such study. For example, we can implement the theoretical results in Chapters 3–5 on the emulator and investigate how they perform. We can also connect real applications to the emulator. Several new challenges arise with running real applications on MMCs, such as: What is the best way of performing service migration and what are the (real) costs associated with it? Service migration is expected to occur more frequently in MMCs than in centralized clouds, because otherwise the benefits of MMCs can no longer be fully realized as users tend to move out of their original area in mobile environments.

Bibliography

Note: The numbers at the end of each reference stand for the page numbers where the reference has been cited.

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. 1

[2] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*. ACM, 2012, pp. 21–28. 1

[3] Y. Abe, R. Geambasu, K. Joshi, H. A. Lagar-Cavilla, and M. Satyanarayanan, "vTube: efficient streaming of virtual appliances over last-mile networks," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 16. 1

[4] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. of ACM MobiSys*, 2014. 1, 2, 7

[5] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," in *Proc. of MobiCASE 2014*, Nov. 2014. 1, 2, 7

[6] "Smarter wireless networks," *IBM Whitepaper No. WSW14201USEN*, Feb. 2013. [Online]. Available: www.ibm.com/services/multimedia/Smarter_wireless_networks.pdf 2

[7] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani, "An open ecosystem for mobile-cloud convergence," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 63–70, Mar. 2015. 2

[8] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, Oct. 2013. 2, 7

[9] S. Davy, J. Famaey, J. Serrat-Fernandez, J. Gorricho, A. Miron, M. Dramitinos, P. Neves, S. Latre, and E. Goshen, "Challenges to support edge-as-a-service," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 132–139, Jan. 2014. 2, 7

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16. 2, 7

[11] Z. Becvar, J. Plachy, and P. Mach, "Path selection using handover in mobile networks with cloud-enabled small cells," in *Proc. of IEEE PIMRC 2014*, Sept. 2014. 2, 7

[12] T. Taleb and A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, Sept. 2013. 2, 7

[13] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan, "The impact of mobile multimedia applications on data center consolidation," in *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, March 2013, pp. 166–176. 2, 7

[14] M. Satyanarayanan, "A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 18, no. 4, pp. 19–23, Jan. 2015. 2, 7

[15] S. Wang, L. Le, N. Zahariev, and K. K. Leung, "Centralized rate control mechanism for cellular-based vehicular networks," in *Proc. of IEEE GLOBECOM 2013*, 2013. 2, 7

[16] A. Fischer, J. Botero, M. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013. 7, 20

[17] M. Chowdhury, M. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, 2012. 7, 24, 25, 26, 55, 56, 151, 169

[18] T. Taleb and A. Ksentini, "An analytical model for follow me cloud," in *Proc. of IEEE GLOBECOM 2013*, Dec. 2013. 7, 65

[19] A. Ksentini, T. Taleb, and M. Chen, "A Markov decision process-based service migration procedure for follow me cloud," in *Proc. of IEEE ICC 2014*, June 2014. 7, 65, 66

[20] IBM CPLEX Optimizer. [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/ 18, 55

[21] OPTI Toolbox. [Online]. Available: http://www.i2c2.aut.ac.nz/Wiki/OPTI/ 18

[22] I. Giurgiu, C. Castillo, A. Tantawi, and M. Steinder, "Enabling efficient placement of virtual infrastructures in the cloud," in *Proceedings of the 13th International Middleware Conference*, ser. Middleware '12, 2012, pp. 332–353. 20

[23] V. V. Vazirani, *Approximation Algorithms*. Springer, 2001. 20, 41, 208

[24] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998. 21, 133

[25] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer, "Minimum congestion mapping in a cloud," in *Proceedings of the 30th annual ACM SIGACT-SIGOPS*

symposium on Principles of distributed computing, ser. PODC '11, 2011, pp. 267–276. 24, 25, 26, 62

[26] D. Dutta, M. Kapralov, I. Post, and R. Shinde, "Embedding paths into trees: VM placement to minimize congestion," in *Algorithms – ESA 2012*, ser. Lecture Notes in Computer Science, L. Epstein and P. Ferragina, Eds. Springer Berlin Heidelberg, 2012, vol. 7501, pp. 431–442. 24, 25, 26

[27] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. of IEEE INFOCOM 2012*, Mar. 2012, pp. 963–971. 25, 26

[28] J.-J. Kuo, H.-H. Yang, and M.-J. Tsai, "Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems," in *Proc. of IEEE INFOCOM 2014*, 2014. 25, 26

[29] R. Hassin, A. Levin, and M. Sviridenko, "Approximating the minimum quadratic assignment problems," *ACM Trans. Algorithms*, vol. 6, no. 1, pp. 18:1–18:10, Dec. 2009. 26

[30] Y.-h. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," *SIGMETRICS Perform. Eval. Rev.*, vol. 28, no. 1, pp. 1–12, June 2000. 27

[31] B. Krishnamachari, D. Estrin, and S. Wicker, "The impact of data aggregation in wireless sensor networks," in *Proc. of 22nd International Conference on Distributed Computing Systems Workshops*, 2002, pp. 575–578. 27

[32] D. Westhoff, J. Girao, and M. Acharya, "Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaptation," *IEEE Trans. on Mobile Computing*, vol. 5, no. 10, pp. 1417–1431, Oct. 2006. 27

[33] IEEE 802.1D, *IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges*, 2004. 27

[34] A. R. Choudhury, S. Das, N. Garg, and A. Kumar, "Rejecting jobs to minimize load and maximum flow-time," in *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Jan. 2015. 32

[35] Y. Azar, "On-line load balancing," *Theoretical Computer Science*, pp. 218–225, 1992. 32

[36] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007. 37, 143

[37] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, "On-line routing of virtual circuits with applications to load balancing and machine scheduling," *J. ACM*, vol. 44, no. 3, pp. 486–504, May 1997. 42, 214

[38] M. Bienkowski, A. Feldmann, J. Grassler, G. Schaffrath, and S. Schmid, "The wide-area virtual service migration problem: A competitive analysis approach," *IEEE/ACM Trans. on Networking*, vol. 22, no. 1, pp. 165–178, Feb. 2014. 65

[39] U. Mandal, M. Habib, S. Zhang, B. Mukherjee, and M. Tornatore, "Greening the cloud using renewable-energy-aware service migration," *IEEE Network*, vol. 27, no. 6, pp. 36–43, Nov. 2013. 65

[40] D. Xenakis, N. Passas, L. Merakos, and C. Verikoukis, "Mobility management for femtocells in LTE-advanced: Key aspects and survey of handover decision algorithms," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 64–91, 2014. 65

[41] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2009, vol. 414. 67, 80, 85, 98, 104, 218, 224, 225

[42] I. Post and Y. Ye, "The simplex method is strongly polynomial for deterministic markov decision processes." in *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013, pp. 1465–1473. 82, 194

[43] H. Qian. (2015, Apr.) Counting the floating point operations (FLOPS). MATLAB Central File Exchange, No. 50608, Ver. 1.0. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/50608 85

[44] S. Elaydi, *An Introduction to Difference Equations, Third Edition*. Springer, 2005. 95

[45] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser, "A parsimonious model of mobile partitioned networks with clustering," in *Proc. of COMSNETS*, Jan. 2009. 125, 171, 185

[46] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD data set epfl/mobility (v. 2009-02-24)," Downloaded from http://crawdad.org/epfl/mobility/, Feb. 2009. 125, 171, 185

[47] L. Kleinrock, *Queuing Systems, Volume II: Computer Applications*. Hoboken, NJ: John Wiley and Sons, 1976. 126

[48] M. Srivatsa, R. Ganti, J. Wang, and V. Kolar, "Map matching: Facts and myths," in *Proc. of ACM SIGSPATIAL 2013*, 2013, pp. 484–487. 132

[49] S. O. Krumke, *Online optimization: Competitive analysis and beyond*. Habilitationsschrift Technische Universitaet Berlin, 2001. 133

[50] Y. Azar, I. R. Cohen, and D. Panigrahi, "Online covering with convex objectives and applications," *CoRR*, vol. abs/1412.3507, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.3507 133

[51] N. Buchbinder, S. Chen, A. Gupta, V. Nagarajan, and J. Naor, "Online packing and covering framework with convex objectives," *CoRR*, vol. abs/1412.8347, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.8347 133

[52] G. Aceto, A. Botta, W. de Donato, and A. Pescape, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093 – 2115, 2013. 140

[53] K. LaCurts, J. Mogul, H. Balakrishnan, and Y. Turner, "Cicada: Introducing predictive guarantees for cloud networks," Jun. 2014. 140

[54] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11, 2011, pp. 1082–1090. 140

[55] B. Korte and J. Vygen, *Combinatorial optimization*. Springer, 2002. 153, 154

[56] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004. 156, 234

[57] J. Ahrenholz. (2010) CORE download. [Online]. Available: http://downloads.pf.itd.nrl.navy.mil/core/ 175

[58] ——, "Comparison of core network emulation platforms," in *Proc. of IEEE MILCOM 2010*, Oct.-Nov. 2010, pp. 166–171. 175

[59] N. Ivanic, B. Rivera, and B. Adamson, "Mobile ad hoc network emulation environment," in *Proc. of IEEE MILCOM 2009*, Oct. 2009. 175

[60] R. Ogier and P. Spagnolo. (2009, Aug.) RFC 5614, Mobile ad hoc network (MANET) extension of OSPF using connected dominating set (CDS) flooding. [Online]. Available: http://www.ietf.org/rfc/rfc5614.txt 178

[61] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. R. Pruhs, and O. Waarts, "On-line load balancing of temporary tasks," *Journal of Algorithms*, vol. 22, no. 1, pp. 93–110, 1997. 193

[62] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, 2010. 196

[63] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," in *Proc. of IFIP Performance 2015*, Oct. 2015. 196

[64] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," *IEEE Trans. on Information Theory*, vol. 59, no. 10, pp. 6301–6327, Oct 2013. 196

[65] M. J. Neely, "Distributed stochastic optimization via correlated scheduling," *IEEE/ACM Trans. on Networking*, 2015, to appear. 196

APPENDIX A

Approximation Ratio for Cycle-free Mapping

In the following, we focus on how well the cycle-free restriction approximates the more general case which allows cycles, for the placement of a single linear application graph. We first show that with the objective of load balancing, the problem of placing a single linear application graph onto a linear physical graph when allowing cycles is NP-hard, and then discuss the approximation ratio of the cycle-free restriction.

Proposition A.1. *The line-to-line placement problem while allowing cycles is NP-hard.*

Proof. The proof is similar to the proof of Proposition 3.2 in Section 3.4.1, namely the problem can be reduced from the minimum makespan scheduling on unrelated parallel machines (MMSUPM) problem. Consider the special case where the edge demand is zero, then the problem is the same with the MMSUPM problem, which deals with placing V jobs onto N machines without restriction on their ordering, with the goal of minimizing the maximum load on each machine. □

To discuss the approximation ratio of the cycle-free assignment, we separately consider edge costs and node costs. The worst case ratio is then the maximum among these two ratios, because we have $\max\{r_1x_1, r_2x_2\} \leq \max\{r_1, r_2\}\max\{x_1, x_2\}$, where $r_1, r_2, x_1, x_2 \geq 0$. The variables x_1 and x_2 can respectively denote the true

optimal maximum costs at nodes and links, and the variables r_1 and r_2 can be their corresponding approximation ratios. Then, $\max\{x_1, x_2\}$ is the true optimal maximum cost when considering nodes and links together, and $\max\{r_1, r_2\}$ is their joint approximation ratio. The joint approximation ratio $\max\{r_1, r_2\}$ is tight when r_1 and r_2 are tight, because we can construct worst-case examples, one with zero node demand and another with zero link demand, and there must exist one worst-case example which has approximation ratio $\max\{r_1, r_2\}$.

The following proposition shows that cycle-free placement is always optimal when only the edge cost is considered.

Proposition A.2. *Cycle-free placement on tree physical graphs always brings lower or equal maximum edge cost compared with placement that allows cycles.*

Proof. Suppose a placement that contains cycles produces a lower maximum edge cost than any cycle-free placement, then there exists v and v_1 $(v_1 > v + 1)$ both placed on a particular node n, while nodes $v+1, ..., v_1-1$ are placed on some nodes among $n+1, ..., N$. In this case, placing nodes $v+1, ..., v_1-1$ all onto node n never increases the maximum edge cost, which shows a contradiction. □

For the node cost, we first consider the case where the physical graph is a single line. We note that in this case the cycle-free placement essentially becomes an "ordered matching", which matches V items into N bins, where the first bin may contain items $1, ..., v_1$, the second bin may contain items $v_1+1, ..., v_2$, and so on. We can also see the problem as partitioning the ordered set $\mathcal{V}$ into N subsets, and each subset contains consecutive elements from $\mathcal{V}$.

Proposition A.3. *When each application node has the same cost when placing it on any physical node, then the cycle-free line-to-line placement brings a tight approximation ratio of 2.*

Proof. Suppose we have V items that can be packed into N bins by an optimal algorithm (which does not impose ordering on items), and the optimal cost at each bin is OPT.

To show that the worst case cost ratio resulting from the ordering cannot be larger than 2, we consider a bin packing where the size of each bin is OPT. (Note that the bin packing problem focuses on minimizing the number of bins with given bin size, which is slightly different from our problem.) Because an optimal solution can pack our V items into N bins with maximum cost OPT, when we are given that the size of each bin is OPT, we can also pack all the V items into N bins. Hence, the optimal solution to the related bin packing problem is N. When we have an ordering, we can do the bin packing by the first-fit algorithm which preserves our ordering. The result of the first-fit algorithm has been proven to be at most $2N$ bins [23].

Now we can combine two neighboring bins into one bin. Because we have at most $2N$ bins from the first-fit algorithm, we will have at most N bins after combination. Also because each bin has size OPT in the bin packing problem, the cost after combination will be at most $2 \cdot \mathrm{OPT}$ for each bin.

This shows that the worst-case cost for ordered items is at most $2 \cdot \mathrm{OPT}$.

To show that the approximation ratio of 2 is tight, we consider the following tight example. Suppose $V = 2N$. Among the $2N$ items, N of them have cost of $(1-\varepsilon)\cdot\mathrm{OPT}$, where $\varepsilon > \frac{1}{1+N}$, the remaining N have cost of $\varepsilon \cdot \mathrm{OPT}$. Obviously, an optimal allocation will put one $(1-\varepsilon)\cdot\mathrm{OPT}$ item and one $\varepsilon\cdot\mathrm{OPT}$ item into one bin, and the resulting maximum cost at each bin is OPT.

A bad ordering could have all $(1-\varepsilon)\cdot\mathrm{OPT}$ items coming first, and all $\varepsilon\cdot\mathrm{OPT}$ items coming afterwards. In this case, if we would like the maximum cost to be smaller than $(2-2\varepsilon)\cdot\mathrm{OPT}$, it would be impossible to fit all the items into N bins, because all the $(1-\varepsilon)\cdot\mathrm{OPT}$ items will already occupy N bins, as it is impossible to put more than one $(1-\varepsilon)\cdot\mathrm{OPT}$ item into each bin if the cost is smaller than

$(2 - 2\varepsilon)\cdot\mathrm{OPT}$, and also because $N\varepsilon\cdot\mathrm{OPT} > \left(\frac{1}{\varepsilon} - 1\right)\varepsilon\cdot\mathrm{OPT} = (1 - \varepsilon)\cdot\mathrm{OPT}$, which means that it is also impossible to put all $\varepsilon\cdot\mathrm{OPT}$ into the last bin on top of the existing $(1 - \varepsilon)\cdot\mathrm{OPT}$ item.

As N becomes arbitrarily large and ε becomes arbitrarily small, we can conclude that the worst-case ordered assignment has at least a cost of $2\cdot\mathrm{OPT}$. □

Corollary A.1. *When the physical graph is a tree, and for each application node, its maximum to minimum cost when placing it on any physical node is $d_{\%,v}$, then the cycle-free line-to-line placement has an approximation ratio of $2V \cdot \max_v d_{\%,v} = O(V)$.*

Proof. This follows from the fact that OPT may choose the minimum cost for each v while the ordered assignment may have to choose the maximum cost for some v, and also, in the worst case, the cycle-free placement may place all application nodes onto one physical node. The factor 2 follows from Proposition A.3. □

It is not straightforward to find out whether the bound in the above corollary is tight or not, thus we do not discuss it here.

We conclude that the cycle-free placement always brings optimal link cost, which is advantageous because the communication bandwidth is often the more scarce resource in cloud environments. The approximation ratio of node costs can be $O(V)$ in some extreme cases. However, the cycle-free restriction is still reasonable in many practical scenarios. Basically, in these scenarios, one cannot split the whole workload onto all the available servers without considering the total link resource consumption. The analysis in this section is also aimed to provide some further insights that helps to justify in what practical scenarios the proposed work is applicable, while further study is worthwhile for some other scenarios.

APPENDIX B

Constant-Plus-Exponential Cost Approximation to General Cost Functions

---

We only focus on the migration cost function $c_m(x)$, because $c_m(x)$ and $c_d(y)$ have the same form. For a given cost function $f(x)$, where $x \geq 0$, we would like to approximate it with the constant-plus-exponential cost function given by $c_m(x) = \beta_c + \beta_l \mu^x$. Note that although we force $c_m(0) = 0$ by definition, we relax that restriction here and only consider the smooth part of the cost function for simplicity. We assume that this smoothness can be extended to $x = 0$, so that $f(x)$ remains smooth at $x = 0$. Under this definition, $f(x)$ may be non-zero at $x = 0$.

Because $c_m(x)$ includes both a constant term and an exponential term, we cannot obtain an exact analytical expression to minimize a commonly used error function, such as the mean squared error. We can, however, solve for the parameters that minimize the error cost function numerically. To reduce the computational complexity compared to a fully numerical solution, we propose an approximate solution in this section.

We are given an integer w that may be chosen according to practical considerations, and we solve for the parameters β_c, β_l, and μ according to the following

system of equations:

$$\beta_c + \beta_l = f(0) \tag{B.1}$$

$$\beta_c + \beta_l \mu^w = f(w) \tag{B.2}$$

$$\beta_c + \beta_l \mu^{2w} = f(2w) \tag{B.3}$$

Subtracting (B.1) respectively from (B.2) and (B.3) and dividing the two results gives

$$\frac{\mu^{2w} - 1}{\mu^w - 1} = \frac{f(2w) - f(0)}{f(w) - f(0)} \tag{B.4}$$

subject to $\mu^w \neq 1$. We can then solve μ^w from (B.4) which gives

$$\mu^w = \frac{R \pm \sqrt{R^2 - 4(R-1)}}{2} \tag{B.5}$$

where $R \triangleq \frac{f(2w)-f(0)}{f(w)-f(0)} \geq 1$. It follows that we always have $R^2 - 4(R-1) \geq 0$ and $\mu^w \geq 0$. To guarantee that $\mu^w \neq 1$, we set $\mu^w = 1 \pm \varepsilon_0$ if $|\mu^w - 1| < \varepsilon_0$, where ε_0 is a small number and the sign is the same as the sign of $\mu^w - 1$. Then, we have

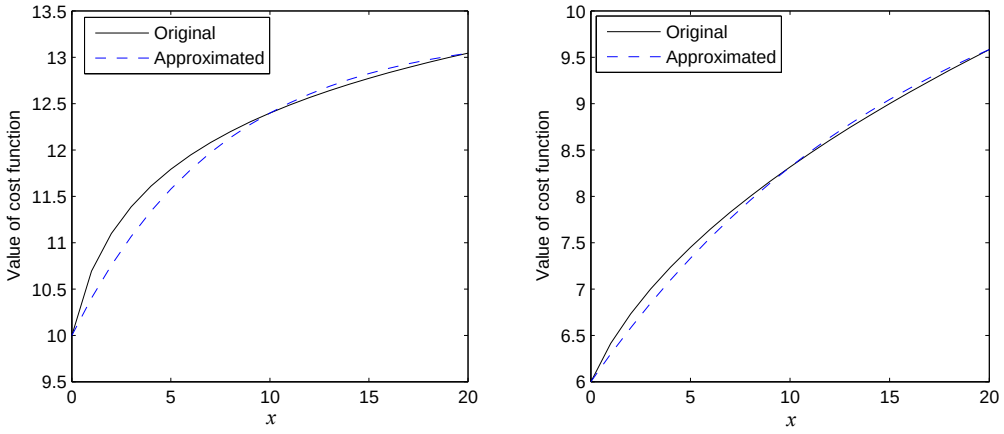
$$\mu = \left(\frac{R \pm \sqrt{R^2 - 4(R-1)}}{2} \right)^{\frac{1}{w}} \tag{B.6}$$

We can then obtain

$$\beta_c = \frac{f(0)\mu^w - f(w)}{\mu^w - 1} \tag{B.7}$$

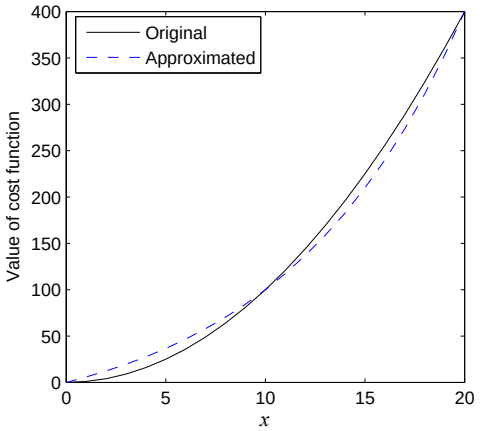
$$\beta_l = \frac{f(w) - f(0)}{\mu^w - 1} \tag{B.8}$$

According to (B.6), we have two possible values of μ, and correspondingly two possible sets of values of β_c and β_l. To determine which is better, we compute the sum squared error $\sum_{x=0}^{2w} \left(f(x) - (\beta_c + \beta_l \mu^x)\right)^2$ for each parameter set, and choose the parameter set that produces the smaller sum squared error.



(a)

(b)



(c)

Figure B.1: Examples of approximating a general cost function with exponential cost function: (a) $f(x) = \ln(x+1) + 10$, (b) $f(x) = \sqrt{x+1} + 5$, (c) $f(x) = x^2$.

Some examples of approximation results are shown in Fig. B.1.

APPENDIX C

Proofs

C.1 Proof of Proposition 3.3

The proof presented here borrows ideas from [37], but is applied here to the generalized case of graph mappings and arbitrary reference offline costs J_{π^o}. For a given $\hat{J}$, we define $\tilde{p}_{n,k}(i) = p_{n,k}(i)/\hat{J}$, $\tilde{d}_{v\to n,k}(i) = d_{v\to n,k}(i)/\hat{J}$, $\tilde{q}_l(i) = q_l(i)/\hat{J}$, and $\tilde{b}_{e\to l}(i) = b_{e\to l}(i)/\hat{J}$. To simplify the proof structure, we first introduce some notations so that the link and node costs can be considered in an identical manner, because it is not necessary to distinguish them in the proof of this proposition. We refer to each type of resources as an *element*, i.e. the type k resource at node n is an element, the resource at link l is also an element. Then, we can define the aggregated cost up to service i for element r as $\tilde{z}_r(i)$. The value of $\tilde{z}_r(i)$ can be either $\tilde{p}_{n,k}(i)$ or $\tilde{q}_l(i)$ depending on the resource type under consideration. Similarly, we define $\tilde{w}_{r|\pi}(i)$ as the incremental cost that service i brings to element r under the mapping π. The value of $\tilde{w}_{r|\pi}(i)$ can be either $\sum_{\forall v:\pi(v)=n} \tilde{d}_{v\to n,k}(i)$ or $\sum_{\forall e=(v_1,v_2):(\pi(v_1),\pi(v_2))\ni l} \tilde{b}_{e\to l}(i)$. Both $\tilde{z}_r(i)$ and $\tilde{w}_{r|\pi}(i)$ are normalized by the reference cost $\hat{J}$.

Using the above notations, the objective function in (3.12) with (3.13a) and (3.13b) becomes

$$\min_{\pi_i} \sum_r \left(\alpha^{\tilde{z}_r(i-1)+\tilde{w}_{r|\pi_i}(i)} - \alpha^{\tilde{z}_r(i-1)}\right). \tag{C.1}$$

Note that due to the notational equivalence, (C.1) is the same as (3.12) with (3.13a)

and (3.13b).

Recall that π^o denotes the reference offline mapping result, let π_i^o denote the offline mapping result for nodes that correspond to the ith service, and $\tilde{z}_r^o(i)$ denote the corresponding aggregated cost until service i. Define the following potential function:

$$\Phi(i) = \sum_r \alpha^{\tilde{z}_r(i)} \left(\gamma - \tilde{z}_r^o(i)\right), \tag{C.2}$$

which helps us prove the proposition. Note that variables without superscript "o" correspond to the values resulting from Algorithm 3.2 that optimizes the objective function (C.1) for each service independently.

The change in $\Phi(i)$ after new service arrival is

$$\begin{aligned}
&\Phi(i) - \Phi(i-1) \\
&= \sum_{r:\exists \pi_i(\cdot)=r} \left(\alpha^{\tilde{z}_r(i)} - \alpha^{\tilde{z}_r(i-1)}\right)\left(\gamma - \tilde{z}_r^o(i-1)\right) - \sum_{r:\exists \pi_i^o(\cdot)=r} \alpha^{\tilde{z}_r(i)} \tilde{w}_{r|\pi_i^o}(i) && \text{(C.3)} \\
&\leq \sum_{r:\exists \pi_i(\cdot)=r} \gamma \left(\alpha^{\tilde{z}_r(i-1)+\tilde{w}_{r|\pi_i}(i)} - \alpha^{\tilde{z}_r(i-1)}\right) - \sum_{r:\exists \pi_i^o(\cdot)=r} \alpha^{\tilde{z}_r(i-1)} \tilde{w}_{r|\pi_i^o}(i) && \text{(C.4)} \\
&\leq \sum_{r:\exists \pi_i^o(\cdot)=r} \gamma \left(\alpha^{\tilde{z}_r(i-1)+\tilde{w}_{r|\pi_i^o}(i)} - \alpha^{\tilde{z}_r(i-1)}\right) - \alpha^{\tilde{z}_r(i-1)} \tilde{w}_{r|\pi_i^o}(i) && \text{(C.5)} \\
&= \sum_{r:\exists \pi_i^o(\cdot)=r} \alpha^{\tilde{z}_r(i-1)} \left\{\gamma \left(\alpha^{\tilde{w}_{r|\pi_i^o}(i)} - 1\right) - \tilde{w}_{r|\pi_i^o}(i)\right\}, && \text{(C.6)}
\end{aligned}$$

where the notation $\pi_i(\cdot) = r$ or $\pi_i^o(\cdot) = r$ means that service i has occupied some resource from element r when respectively using the mapping from Algorithm 3.2 or the reference offline mapping.

We explain the relationships in (C.3) to (C.6) in the following. Equality (C.3) follows from

$$\begin{aligned}
&\Phi(i) - \Phi(i-1) \\
&= \sum_r \alpha^{\tilde{z}_r(i)} \left(\gamma - \left(\tilde{z}_r^o(i-1) + \tilde{w}_{r|\pi_i^o}(i)\right)\right) - \sum_r \alpha^{\tilde{z}_r(i-1)} \left(\gamma - \tilde{z}_r^o(i-1)\right)
\end{aligned}$$

$$= \sum_r \left(\alpha^{\tilde{z}_r(i)} - \alpha^{\tilde{z}_r(i-1)}\right)(\gamma - \tilde{z}_r^o(i-1)) - \sum_r \alpha^{\tilde{z}_r(i)} \tilde{w}_{r|\pi_i^o}(i)$$

$$= \sum_{r:\exists \pi_i(\cdot)=r} \left(\alpha^{\tilde{z}_r(i)} - \alpha^{\tilde{z}_r(i-1)}\right)(\gamma - \tilde{z}_r^o(i-1)) - \sum_{r:\exists \pi_i^o(\cdot)=r} \alpha^{\tilde{z}_r(i)} \tilde{w}_{r|\pi_i^o}(i),$$

where the last equality follows from the fact that $\alpha^{\tilde{z}_r(i)} - \alpha^{\tilde{z}_r(i-1)} = 0$ for all r such that $\forall \pi_i(\cdot) \neq r$, and $\tilde{w}_{r|\pi_i^o}(i) = 0$ for all r such that $\forall \pi_i^o(\cdot) \neq r$. Inequality (C.4) follows from $\tilde{z}_r^o(i-1) \geq 0$ and $\tilde{z}_r(i) = \tilde{z}_r(i-1) + \tilde{w}_{r|\pi_i}(i)$. Note that the first term in (C.4) is the same as the objective function (C.1). Because the mapping π_i results from Algorithm 3.2 which optimizes (C.1), we know that the reference mapping π_0 must produce a cost $\alpha^{\tilde{z}_r(i-1)+\tilde{w}_{r|\pi_i^o}(i)} - \alpha^{\tilde{z}_r(i-1)}$ that is greater than or equal to the optimum, hence following (C.5). Equality (C.6) is obvious.

Now we proof that the potential function $\Phi(i)$ does not increase with i, by proving that (C.6) is not larger than zero. For the ith request, the reference offline mapping produces the mapping result π_i^o. Therefore, for all r such that $\exists \pi_i^o(\cdot) = r$, we have $0 \leq \tilde{w}_{r|\pi_i^o}(i) \leq J_{\pi^o}/\hat{J} \leq 1$. Hence, we only need to show that $\gamma\left(\alpha^{\tilde{w}_{r|\pi_i^o}(i)} - 1\right) - \tilde{w}_{r|\pi_i^o}(i) \leq 0$ for $\tilde{w}_{r|\pi_i^o}(i) \in [0,1]$, which is true for $\alpha \leq 1 + 1/\gamma$. From (C.3)–(C.6), it follows that $\Phi(i) \leq \Phi(i-1)$. (We take $\alpha = 1 + 1/\gamma$ because this results in the smallest value of β.)

Because $\tilde{z}_r(0) = \tilde{z}_r^o(0) = 0$, we have $\Phi(0) = \gamma(NK + L)$. Because $\Phi(i)$ does not increase, $\alpha > 1$, and $\tilde{z}_r^o(i) \leq 1$ due to $J_{\pi^o} \leq \hat{J}$, we have

$$(\gamma - 1)\,\alpha^{\max_r \tilde{z}_r(i)} \leq (\gamma - 1)\sum_r \alpha^{\tilde{z}_r(i)} \leq \Phi(i) \leq \Phi(0) = \gamma(NK + L). \quad \text{(C.7)}$$

Taking the logarithm on the left and right side of (C.7), we have

$$\max_r \tilde{z}_r(i) \leq \log_\alpha \left(\frac{\gamma(NK + L)}{\gamma - 1}\right) = \beta, \quad \text{(C.8)}$$

which proves the result because $z_r(i) = \tilde{z}_r(i) \cdot \hat{J}$.

C.2 Proofs of Proposition 4.1 and Corollary 4.1

C.2.1 Proof of Proposition 4.1

Suppose that we are given a service migration policy π such that the service can be migrated to a location that is farther away from the user, i.e., $\|u-h'\| > \|u-h\|$. We will show that, for an arbitrary sample path of the user locations $u(t)$, we can find a (possibly history dependent) policy ψ that does not migrate to locations farther away from the user in any timeslot, which performs not worse than policy π.

For an arbitrary sample path of user locations $u(t)$, denote the timeslot t_0 as the *first* timeslot (starting from $t = 0$) in which the service is migrated to somewhere farther away from the user when following policy π. The initial state at timeslot t_0 is denoted by $s(t_0) = (u(t_0), h(t_0))$. When following π, the state shifts from $s(t_0)$ to $s'_\pi(t_0) = (u(t_0), h'_\pi(t_0))$, where $\|u(t_0)-h'_\pi(t_0)\| > \|u(t_0)-h(t_0)\|$. The subsequent states in this case are denoted by $s_\pi(t) = (u(t), h_\pi(t))$ for $t > t_0$.

Now, we define a policy ψ such that the following conditions are satisfied for the given sample path of user locations $u(t)$:

- The migration actions in timeslots $t < t_0$ are the same when following either ψ or π.
- The policy ψ specifies that there is no migration within the timeslots $t \in [t_0, t_m - 1]$, where $t_m > t_0$ is a timeslot index that is defined later.
- The policy ψ is defined such that, at timeslot t_m, the service is migrated to $h'_\pi(t_m)$, where $h'_\pi(t_m)$ is the service location (after possible migration at timeslot t_m) when following π.
- For timeslots $t > t_m$, the migration actions for policies ψ and π are the same.

For $t > t_0$, the states when following ψ are denoted by $s_\psi(t) = (u(t), h_\psi(t))$.

The timeslot t_m is defined as the *first* timeslot after t_0 such that the following condition is satisfied:

1. $\|u(t_m) - h_\psi(t_m)\| > \|u(t_m) - h'_\pi(t_m)\|$, i.e., the transmission cost (before migration at timeslot t_m) when following ψ is larger than the transmission cost (after possible migration at timeslot t_m) when following π.

Accordingly, within the interval $[t_0 + 1, t_m - 1]$, the transmission cost when following ψ is always less than or equal to the transmission cost when following π, and there is no migration when following ψ. Therefore, for timeslots $t \in [t_0, t_m - 1]$, policy π cannot bring lower cost than policy ψ.

In timeslot t_m, we can always choose a migration action for ψ where the migration cost is smaller than or equal to the sum of the migration costs of π within $[t_0, t_m]$. The reason is that ψ can migrate (in timeslot t_m) following the same migration path as π within $[t_0, t_m]$.

It follows that, for timeslots within $[t_0, t_m]$, policy π cannot perform better than policy ψ, and both policies have the same costs for timeslots within $[0, t_0 - 1]$ and $[t_m + 1, \infty)$. The above procedure can be repeated so that all the migration actions to a location farther away from the user can be removed without increasing the overall cost, thus we can redefine ψ to be a policy that removes all such migrations.

We note that the policy ψ can be constructed based on policy π without prior knowledge of the user's future locations. For any policy π, the policy ψ is a policy that does not migrate whenever π migrates to a location farther away from the user (corresponding to timeslot t_0). Then, it migrates to $h'_\pi(t_m)$ when condition 1 is satisfied (this is the timeslot t_m in the above discussion).

The policy ψ is a history dependent policy, because its actions depend on the past actions of the underlying policy π. From [41, Chapter 6], we know that history dependent policies cannot outperform Markovian policies for our problem, assuming that the action spaces of both policies are identical for every possible state. There-

fore, there exists a Markovian policy that does not migrate to a location farther away from the user, which does not perform worse than π. Noting that the optimal policy found from (4.2) and (4.3) are Markovian policies, we have proved the proposition.

C.2.2 Proof of Corollary 4.1

The proof follows the same procedure as the proof of Proposition 4.1. For any given policy π that migrates to locations other than the user location, we show that there exists a policy ψ that does not perform such migration, which performs not worse than the original policy π. The difference is that t_0 is defined as the first timeslot such that $u(t_0) \neq h'_\pi(t_0)$, and t_m is defined as the first timeslot after t_0 such that $u(t_m) = h'_\pi(t_m)$. Due to the strict inequality relationship of the cost functions given in the corollary, and because $0 < \gamma < 1$, we can conclude that π is not optimal.

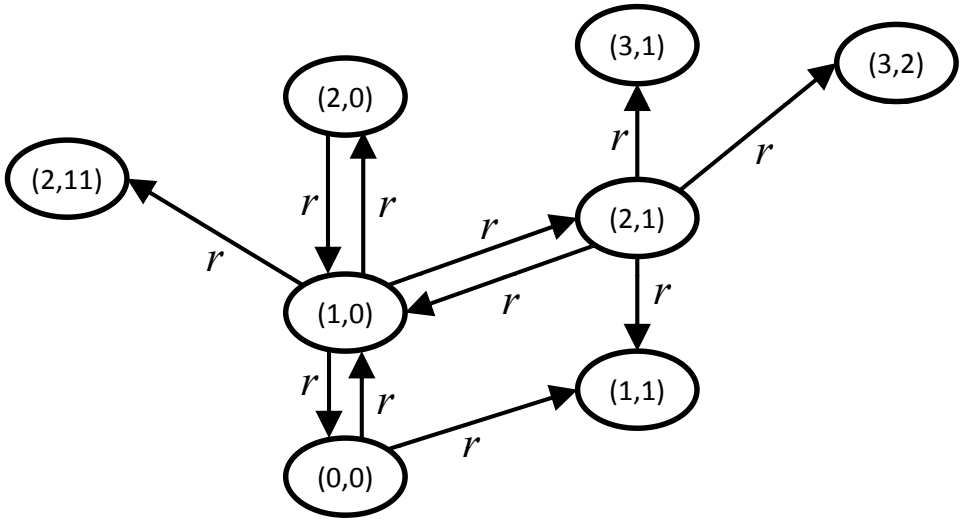
C.3 Proof of Proposition 4.5

Recall that among the neighbors (i', j') of a cell (i, j) in the 2-D offset-based MDP $\{e(t)\}$, when $i > 0$, we have two (or, correspondingly, one) cells with $i' = i - 1$, and two (or, correspondingly, three) cells with $i' = i + 1$. To "even out" the different number of neighboring cells, we define a new (modified) MDP $\{g(t)\}$ for the 2-D offset-based model, where the states are connected as in the original 2-D MDP, but the transition probabilities are different, as shown in Fig. C.1.

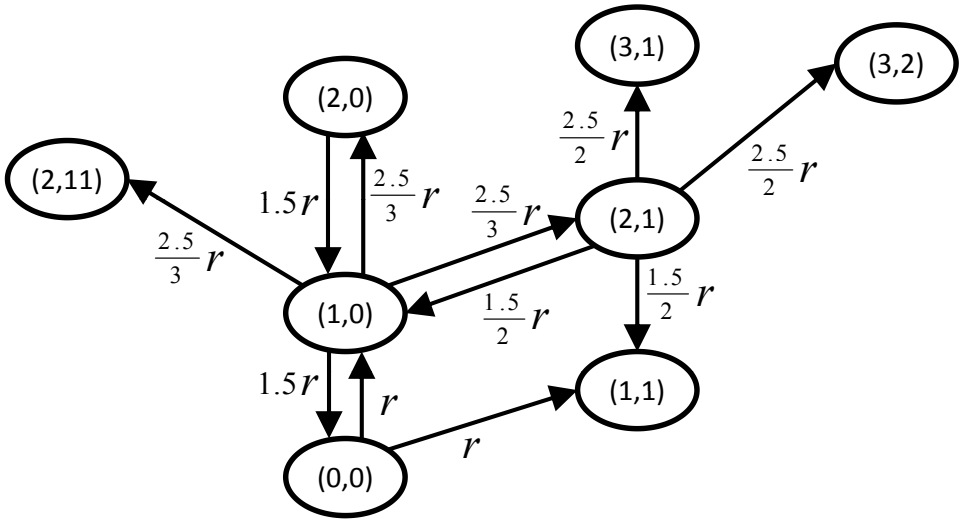
In the modified MDP $\{g(t)\}$, the transition probabilities starting from state $(0, 0)$ to each of its neighboring cells have the same value r. For all the other states $(i, j) \neq (0, 0)$, they are defined as follows:

- The transition probability to each of its neighbors with the same ring index i is r.
- If state (i, j) has two (correspondingly, one) neighbors in the lower ring $i - 1$, then the transition probability to each of its neighbors in the lower ring is $\frac{1.5}{2}r$ (correspondingly, $1.5r$).
- if state (i, j) has two (correspondingly, three) neighbors in the higher ring $i+1$, then the transition probability to each of its neighbors in the higher ring is $\frac{2.5}{2}r$ (correspondingly, $\frac{2.5}{3}r$).

We denote the discounted sum cost from the original MDP $\{e(t)\}$ by $V(i, j)$, and denote that from the modified MDP $\{g(t)\}$ by $U(i, j)$, where (i, j) stands for the initial state in the discounted sum cost definition.



(a)



(b)

Figure C.1: Illustration of original and modified 2-D MDPs, only some exemplar states and transition probabilities are shown: (a) original, (b) modified.

Part I – Upper bound on the difference between $V(i,j)$ and $U(i,j)$ for a given policy π

Assume we have the same policy π for the original and modified MDPs. Then, the balance equation of $V(i,j)$ is

$$V(i,j) = C_a(i,j) + \gamma \left((1-6r)\, V(a(i,j)) + r \sum_{(i',j') \in \mathcal{N}(a(i,j))} V(i',j') \right) \tag{C.9}$$

where $a(i,j)$ is the new state after possible migration at state (i,j), and $\mathcal{N}\left(a(i,j)\right)$ is the set of states that are neighbors of state $a(i,j)$.

For $U(i,j)$, we have

$$\begin{aligned} U(i,j) = C_a(i,j) + \gamma \Bigg(& (1-6r)\, U(a(i,j)) \\ & + \sum_{(i',j') \in \mathcal{N}(a(i,j))} P_{g'(t)=a(i,j),g(t+1)=(i',j')} U(i',j') \Bigg) \end{aligned} \tag{C.10}$$

where $P_{g'(t)=a(i,j),g(t+1)=(i',j')}$ is the transition probability of the modified MDP $\{g(t)\}$ as specified earlier.

In the following, let $i_{a(i,j)}$ denote the ring index of $a(i,j)$. We define sets

$$\begin{aligned} \mathcal{N}^{-}(a(i,j)) &= \left\{ (i',j') \in \mathcal{N}\left(a(i,j)\right) : i' = i_{a(i,j)} - 1 \right\} \\ \mathcal{N}^{+}(a(i,j)) &= \left\{ (i',j') \in \mathcal{N}\left(a(i,j)\right) : i' = i_{a(i,j)} + 1 \right\} \end{aligned}$$

to represent the neighboring states of $a(i,j)$ that are respectively in the lower and higher rings. We use $|\cdot|$ to denote the number of elements in a set.

Assume $|U(i,j) - V(i,j)| \leq \varepsilon$ for all i and j, and the value of ε is unknown for now. We subtract (C.9) from (C.10), and then take the absolute value, yielding (C.11), (C.12), and (C.13) which are explained below, where the set $\mathcal{N}_2\left(i,j\right)$ is the

$$
\begin{aligned}
&|U(i,j) - V(i,j)| \\
&= \gamma \Big| (1-6r)\left(U(a(i,j)) - V(a(i,j))\right) \\
&\quad + \sum_{(i',j') \in \mathcal{N}(a(i,j))} P_{g'(t)=a(i,j),g(t+1)=(i',j')} \left(U(i',j') - V(i',j')\right) \\
&\quad \pm 0.5\lambda_{i_{a(i,j)}} r \left(\frac{\sum_{(i',j') \in \mathcal{N}^+(a(i,j))} V(i',j')}{|\mathcal{N}^+(a(i,j))|} - \frac{\sum_{(i',j') \in \mathcal{N}^-(a(i,j))} V(i',j')}{|\mathcal{N}^-(a(i,j))|} \right) \Big| \qquad \text{(C.11)} \\
&\leq \gamma\varepsilon + 0.5\gamma r \left| \frac{\sum_{(i',j') \in \mathcal{N}^+(a(i,j))} V(i',j')}{|\mathcal{N}^+(a(i,j))|} - \frac{\sum_{(i',j') \in \mathcal{N}^-(a(i,j))} V(i',j')}{|\mathcal{N}^-(a(i,j))|} \right| \qquad \text{(C.12)} \\
&\leq \gamma\varepsilon + 0.5\gamma r \max_{i,j,j': (i+1,j) \in \mathcal{N}_2(i-1,j')} |(V(i+1,j) - V(i-1,j'))| \qquad \text{(C.13)}
\end{aligned}
$$

---

set of states that are two-hop neighbors of state (i,j), the variable $\lambda_{i_{a(i,j)}} = 0$ when $i_{a(i,j)} = 0$, and $\lambda_{i_{a(i,j)}} = 1$ when $i_{a(i,j)} > 0$.

The first two terms of (C.11) subtract the discounted sum cost of the original MDP $\{e(t)\}$ from that of the modified MDP $\{g(t)\}$, by assuming that both chains have the *same* transition probabilities specified by the modified MDP. The difference in their transition proabilities is captured by the last term of (C.11). There is no difference in the transition probabilities when $i_{a(i,j)} = 0$, thus $\lambda_{i_{a(i,j)}} = 0$ when $i_{a(i,j)} = 0$.

In the following, we consider $i_{a(i,j)} > 0$ and further explain the last term of (C.11). We first note that there is difference in the transition probabilities only when moving to the lower or higher ring:

- The *sum* probability of moving to the lower ring in $\{g(t)\}$ is by $0.5r$ smaller (or, correspondingly, greater) than that in $\{e(t)\}$.
- The *sum* probability of moving to the higher ring in $\{g(t)\}$ is by $0.5r$ greater (or, correspondingly, smaller) than that in $\{e(t)\}$.

Therefore, the transition probablity difference for each neighboring state in the lower

(or higher) ring is $\pm 0.5r$ divided by the number of neighbors in the lower (or higher) ring. Also note that the probablity difference for lower and higher rings have opposite signs. This explains the last term of (C.11), which captures the difference in the transition probabilities and its impact on the discounted sum costs.

The inequality in (C.12) is from the triangle inequality. We note that the subtraction in the last term of (C.11) only occurs on $V(i', j')$ values that are two-hop neighbors, so we have the inequality in (C.13) by replacing the value with the maximum.

From (C.13), we can obtain a balance equation for the upper bound of $|U(i,j) - V(i,j)|$, which is

$$\varepsilon = \gamma\varepsilon + 0.5\gamma r \max_{i,j,j':(i+1,j)\in\mathcal{N}_2(i-1,j')} |(V(i+1,j) - V(i-1,j'))| \tag{C.14}$$

Because $0 < \gamma < 1$, the value of $V(i,j)$ converges after a number of iterations according to (C.9) [41, Chapter 6], so $|(V(i+1,j) - V(i-1,j))|$ also converges, and the value of ε can be solved by

$$\varepsilon_V = \frac{\gamma r \max_{i,j,j':(i+1,j)\in\mathcal{N}_2(i-1,j')} |(V(i+1,j) - V(i-1,j'))|}{2(1-\gamma)} \tag{C.15}$$

Note that the above argument also applies when interchanging V and U, which means that an alternative upper bound of the cost is

$$\varepsilon_U = \frac{\gamma r \max_{i,j,j':(i+1,j)\in\mathcal{N}_2(i-1,j')} |(U(i+1,j) - U(i-1,j'))|}{2(1-\gamma)} \tag{C.16}$$

The upper bound can also be expressed as $\varepsilon = \min\{\varepsilon_V, \varepsilon_U\}$, but either ε_V or ε_U may have the smaller value.

Part II – Optimal policy for the modified 2-D MDP $\{g(t)\}$ is equivalent to the optimal policy for the distance-based MDP $\{d(t)\}$

We note that the optimal policy of an MDP can be found from value iteration [41, Chapter 6]. For the modified MDP $\{g(t)\}$, we initialize with a never migrate policy, which gives the initial value function $U_0(i,j) = c_d(i)$, satisfying $U_0(i,j) = U_0(i,j')$ for all i and $j \neq j'$.

Suppose $U_n(i,j) = U_n(i,j')$ for all i and $j \neq j'$. In each iteration, we use the following equation to obtain the new value function and the corresponding actions for each i and j:

$$U_{n+1}(i,j) = \min_a \Bigg\{ C_a(i,j) + \gamma \sum_{i'} \sum_{j'} P_{g'(t)=a(i,j),g(t+1)=(i',j')} U_n(i',j') \Bigg\} \tag{C.17}$$

From the hexagon model in Fig. 4.9, we can see that for ring indices i and i', where $i' < i$, we can always reach from state (i,j) to a state in ring i' with $i - i'$ hops, regardless of the index j. In the $(n+1)$th iteration, if it is optimal to migrate from state (i,j) to a state with ring index i', then the migration destination must be $i - i'$ hops away from origin state, because $U_n(i',j) = U_n(i',j')$ for all i' and $j \neq j'$, it cannot be beneficial to migrate to somewhere farther away.

Further, if it is optimal to migrate at a state (i,j) to a state in ring i', it must be optimal to migrate at states (i,j) for all j to a state (which may not be the same state) in ring i', bringing the same cost, i.e. $U_{n+1}(i,j) = U_{n+1}(i,j')$ for $j \neq j'$. This is due to the symmetry of cost functions (in the sense that $U_n(i,j) = U_n(i,j')$ for all i and $j \neq j'$) and symmetry of transition probabilities (in the sense that the sum probability of reaching ring $i-1$ from any state in ring i is the same, and the sum probability of reaching ring $i+1$ from any state in ring i is also the same). Similarly,

if it is optimal not to migrate at a state (i, j), then it is optimal not to migrate at states (i, j) for all j, which also brings $U_{n+1}(i, j) = U_{n+1}(i, j')$ for any $j \neq j'$.

Because the value iteration converges to the optimal policy and its corresponding cost as $n \to \infty$, for the optimal policy of the modified MDP $\{g(t)\}$, we have the same discounted sum cost for states in the same ring, i.e. $U^*(i, j) = U^*(i, j')$ for all i and $j \neq j'$. Meanwhile, for a given i, the optimal actions $a^*(i, j)$ and $a^*(i, j')$ for any $j \neq j'$ have the same ring index $i_{a^*(i,j)}$.

Since the optimal actions $a^*(i, j)$ only depend on the ring index i and the ring index of $a^*(i, j)$ does not change with j, the optimal policy for $\{g(t)\}$ can be directly mapped to a policy for the distance-based MDP $\{d(t)\}$. A policy for $\{d(t)\}$ can also be mapped to a policy for $\{g(t)\}$ by considering the shortest path between different states in $\{g(t)\}$, as discussed in Section 4.4.3.2. This implies that there is a one-to-one mapping between the optimal policy for $\{g(t)\}$ and a policy for $\{d(t)\}$, because the optimal policy for $\{g(t)\}$ also only migrates along the shortest path between states.

We now show that the policy for $\{d(t)\}$ obtained from the optimal policy for $\{g(t)\}$ is optimal for $\{d(t)\}$. To find the optimal policy for $\{d(t)\}$, we can perform value iteration according to the following update equation:

$$U_{n+1}(i) = \min_a \Bigg\{ C_a(i) + \gamma \sum_{i'} \left(\sum_{j'} P_{g(t)=a(i),g(t+1)=(i',j')} \right) U_n(i') \Bigg\} \tag{C.18}$$

The difference between (C.17) and (C.18) is that (C.18) does not distinguish the actions and value functions with different j indices. Recall that for the modified 2-D MDP $\{g(t)\}$, we have $U_n(i, j) = U_n(i, j')$ for all n, i and $j \neq j'$, so the index j can be natually removed from the value functions. Further, if it is optimal to migrate at state (i, j) to a state in ring i', it must be optimal to migrate at states (i, j) for all j to

a state (which may not be the same state) in ring i'. The migration cost for different j are the same because they all follow the shortest path from state (i, j) to ring i'. If it is optimal not to migrate at state (i, j), then it is optimal not to migrate at states (i, j) for all j. Therefore, we can also remove the j index associated with the actions, without affecting the value function. It follows that the optimal policy for $\{g(t)\}$ is equivalent to the optimal policy for $\{d(t)\}$, both bringing the same value functions (discounted sum costs).

Part III – Error bound for distance-based approximation

By now, we have shown the upper bound on the discounted sum cost difference between the original and modified 2-D MDPs $\{e(t)\}$ and $\{g(t)\}$, when both MDPs use the same policy. We have also shown that the optimal policy for the modified 2-D MDP $\{g(t)\}$ is equivalent to the optimal policy for the distance-based MDP $\{d(t)\}$. Note that the true optimal cost is obtained by solving for the optimal policy for the original 2-D MDP $\{e(t)\}$, and the approximate optimal cost is obtained by applying the optimal policy for $\{d(t)\}$ to $\{e(t)\}$. In the following, we consider the upper bound on the difference between the true and approximate optimal discounted sum costs.

We start with a (true) optimal policy π^*_{true} for $\{e(t)\}$, denote the discounted sum costs from this policy as $V_{\pi^*_{\text{true}}}(i, j)$. When using the same policy on $\{g(t)\}$, the difference between the costs $U_{\pi^*_{\text{true}}}(i, j)$ and $V_{\pi^*_{\text{true}}}(i, j)$ satisfies the upper bound given in (C.15), i.e.

$$U_{\pi^*_{\text{true}}}(i, j) - V_{\pi^*_{\text{true}}}(i, j) \leq \varepsilon_{V_{\pi^*_{\text{true}}}} \tag{C.19}$$

Since $V_{\pi^*_{\text{true}}}(i, j)$ are the optimal costs, we have

$$\max_{i,j,j':(i+1,j)\in\mathcal{N}_2(i-1,j')} \left|\left(V_{\pi^*_{\text{true}}}(i+1, j) - V_{\pi^*_{\text{true}}}(i-1, j')\right)\right|$$

$$\leq \max_x \{c_m(x+2) - c_m(x)\} \tag{C.20}$$

because, otherwise, there exists at least one pair of states $(i+1, j)$ and $(i-1, j')$ for which it is beneficial to migrate from state $(i+1, j)$ to state $(i-1, j')$, according to a 2-D extension of (4.29). The cost after performing such migration is upper bounded by (C.20), which contradicts with the fact that π^*_{true} is optimal.

Define

$$\varepsilon_c = \frac{\gamma r \max_x \{c_m(x+2) - c_m(x)\}}{2(1-\gamma)} \tag{C.21}$$

From (C.15), (C.19) and (C.20), we have

$$U_{\pi^*_{\text{true}}}(i,j) - V_{\pi^*_{\text{true}}}(i,j) \leq \varepsilon_c \tag{C.22}$$

According to the equivalence of $\{g(t)\}$ and $\{d(t)\}$, we know that the optimal policy π^*_{appr} of $\{d(t)\}$ is also optimal for $\{g(t)\}$. Hence, we have

$$U_{\pi^*_{\text{appr}}}(i,j) \leq U_{\pi^*_{\text{true}}}(i,j) \tag{C.23}$$

because the cost from the optimal policy cannot be higher than the cost from any other policy.

When using the policy π^*_{appr} on $\{e(t)\}$, we get costs $V_{\pi^*_{\text{appr}}}(i,j)$. From (C.16), and because $U_{\pi^*_{\text{appr}}}(i,j)$ is the optimal cost for $\{g(t)\}$, we have

$$V_{\pi^*_{\text{appr}}}(i,j) - U_{\pi^*_{\text{appr}}}(i,j) \leq \varepsilon_{U_{\pi^*_{\text{appr}}}} \leq \varepsilon_c \tag{C.24}$$

From (C.22), (C.23), and (C.24), we get

$$V_{\pi^*_{\text{appr}}}(i,j) - V_{\pi^*_{\text{true}}}(i,j) \leq 2\varepsilon_c \tag{C.25}$$

which completes the proof.

C.4 Proof of Proposition 5.2

We first introduce a few lemmas, with results used later in the proof.

Lemma C.1. *For any instance j and configuration sequence $\boldsymbol{\lambda}$, we have*

$$\frac{\partial \widetilde{D}}{\partial x_{j\boldsymbol{\lambda}}}(\mathbf{x}) = \nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}(\mathbf{y},\mathbf{z}) \cdot (\mathbf{a}_{j\boldsymbol{\lambda}}, \mathbf{b}_{j\boldsymbol{\lambda}}) \tag{C.26}$$

Proof.

$$\begin{aligned}
\frac{\partial \widetilde{D}}{\partial x_{j\boldsymbol{\lambda}}}(\mathbf{x}) &= \sum_{t=t_0}^{t_0+T-1} \left[\sum_{n=1}^{N} \frac{\partial \widetilde{D}}{\partial y_n(t)}(\mathbf{x}) \cdot \frac{\partial y_n(t)}{\partial x_{j\boldsymbol{\lambda}}}(\mathbf{x}) + \right. \\
&\qquad \left. \sum_{n=1}^{N}\sum_{h=1}^{N} \frac{\partial \widetilde{D}}{\partial z_{nh}(t)}(\mathbf{x}) \cdot \frac{\partial z_{nh}(t)}{\partial x_{j\boldsymbol{\lambda}}}(\mathbf{x}) \right] \\
&= \sum_{t=t_0}^{t_0+T-1} \left[\sum_{n=1}^{N} \frac{\partial \widetilde{D}}{\partial y_n(t)}(\mathbf{x}) \cdot a_{j\boldsymbol{\lambda}n}(t) + \right. \\
&\qquad \left. \sum_{n=1}^{N}\sum_{h=1}^{N} \frac{\partial \widetilde{D}}{\partial z_{nh}(t)}(\mathbf{x}) \cdot b_{j\boldsymbol{\lambda}nh}(t) \right] \\
&= \nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}(\mathbf{y},\mathbf{z}) \cdot (\mathbf{a}_{j\boldsymbol{\lambda}}, \mathbf{b}_{j\boldsymbol{\lambda}})
\end{aligned}$$

where we recall that $y_n(t)$ and $z_{nh}(t)$ are functions of $x_{j\boldsymbol{\lambda}}$ for all j and $\boldsymbol{\lambda}$, thus they are also functions of vector $\mathbf{x}$. □

Lemma C.2. *For any instance j and configuration sequence $\boldsymbol{\lambda}$, we have*

$$\nabla_{\mathbf{x}} \widetilde{D}(\mathbf{x}) \cdot \mathbf{x} = \nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}(\mathbf{y},\mathbf{z}) \cdot (\mathbf{y},\mathbf{z}) \tag{C.27}$$

Proof.

$$\begin{aligned}
&\nabla_{\mathbf{x}} \widetilde{D}(\mathbf{x}) \cdot \mathbf{x} \\
&= \sum_{j=1}^{M} \sum_{\boldsymbol{\lambda}\in\Lambda} \frac{\partial \widetilde{D}}{\partial x_{j\boldsymbol{\lambda}}}(\mathbf{x}) \cdot x_{j\boldsymbol{\lambda}}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=1}^{M} \sum_{\boldsymbol{\lambda} \in \Lambda} \nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}\left(\mathbf{y}, \mathbf{z}\right) \cdot \left(\mathbf{a}_{j\boldsymbol{\lambda}}, \mathbf{b}_{j\boldsymbol{\lambda}}\right) \cdot x_{j\boldsymbol{\lambda}} \\
&= \nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}\left(\mathbf{y}, \mathbf{z}\right) \cdot \left(\sum_{j=1}^{M} \sum_{\boldsymbol{\lambda} \in \Lambda} \left(\mathbf{a}_{j\boldsymbol{\lambda}}, \mathbf{b}_{j\boldsymbol{\lambda}}\right) \cdot x_{j\boldsymbol{\lambda}} \right) \\
&= \nabla_{\mathbf{y},\mathbf{z}} \widetilde{D}\left(\mathbf{y}, \mathbf{z}\right) \cdot \left(\mathbf{y}, \mathbf{z}\right)
\end{aligned}
$$

where the second step follows from Lemma C.1, the last step follows from the definition of vectors $\mathbf{y}, \mathbf{z}, \mathbf{a}_{j\boldsymbol{\lambda}}, \mathbf{b}_{j\boldsymbol{\lambda}}$. □

We introduce some additional notations that are used in the proof below. Recall that the values of vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ may vary over time due to service arrivals and departures. Let $\mathbf{x}_j^{(j)}$, $\mathbf{y}_j^{(j)}$, and $\mathbf{z}_j^{(j)}$ respectively denote the values of $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ *immediately after* instance j is placed; and let $\mathbf{x}_{j-1}^{(j)}$, $\mathbf{y}_{j-1}^{(j)}$, and $\mathbf{z}_{j-1}^{(j)}$ respectively denote the values of $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ *immediately before* instance j is placed. We note that the values of $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ may change after placing each instance. Therefore, the notions of "before", "after", and "time" (used below) here correspond to the sequence of service instance placement, instead of the actual physical time.

We then introduce vectors that only consider the placement up to the jth service instance, which are necessary because the proof below uses an iterative approach. Let $\mathbf{x}_j$, $\mathbf{y}_j$, and $\mathbf{z}_j$ respectively denote the values of $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ at *any time after* placing instance j (where instance j can be either still running in the system or already departed) while *ignoring the placement of any subsequent instances* $j' > j$ (if any). This means, in vector $\mathbf{x}_j$, we set $\left(\mathbf{x}_j\right)_{i\boldsymbol{\lambda}} \triangleq x_{i\boldsymbol{\lambda}}$ for any $i \leq j$ and $\boldsymbol{\lambda}$, and set $\left(\mathbf{x}_j\right)_{i\boldsymbol{\lambda}} \triangleq 0$ for any $i > j$ and $\boldsymbol{\lambda}$, although the value of $x_{i\boldsymbol{\lambda}}$ at the current time of interest may be non-zero for some $i > j$ and $\boldsymbol{\lambda}$. Similarly, in vectors $\mathbf{y}_j$ and $\mathbf{z}_j$, we only consider the resource consumptions up to instance j, i.e., $\left(\mathbf{y}_j\right)_{nt} \triangleq \sum_{i=1}^{j} \sum_{\boldsymbol{\lambda} \in \Lambda} a_{i\boldsymbol{\lambda}n}(t) x_{i\boldsymbol{\lambda}}$ and $\left(\mathbf{z}_j\right)_{nht} \triangleq \sum_{i=1}^{j} \sum_{\boldsymbol{\lambda} \in \Lambda} b_{i\boldsymbol{\lambda}nh}(t) x_{i\boldsymbol{\lambda}}$ for any n, h, and t.

We assume that the last service instance that has arrived before the current time

of interest has index M, thus $\mathbf{x} = \mathbf{x}_M$, $\mathbf{y} = \mathbf{y}_M$, and $\mathbf{z} = \mathbf{z}_M$.

Because an instance will never come back after it has departed (even if an instance of the same type comes back, it will be given a new index), we have $\mathbf{y}_{j-1} \leq \mathbf{y}_{j-1}^{(j)}$ and $\mathbf{z}_{j-1} \leq \mathbf{z}_{j-1}^{(j)}$, where the inequalities are defined element-wise for the vector.

Define $v_j \triangleq \widetilde{D}\left(\mathbf{y}_j^{(j)}, \mathbf{z}_j^{(j)}\right) - \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}, \mathbf{z}_{j-1}^{(j)}\right)$ to denote the increase in the sum cost $\widetilde{D}(\mathbf{y}, \mathbf{z})$ (or equivalently, $\widetilde{D}(\mathbf{x})$) at the time when placing service j. Note that after this placement, the value of $\widetilde{D}\left(\mathbf{y}_j, \mathbf{z}_j\right) - \widetilde{D}\left(\mathbf{y}_{j-1}, \mathbf{z}_{j-1}\right)$ may vary over time, because some services $i \leq j$ may leave the system, but the value of v_j is only taken when service j is placed upon its arrival.

Lemma C.3. *When Assumption 5.1 is satisfied, for any M, we have*

$$\widetilde{D}\left(\mathbf{x}_M\right) \leq \sum_{j=1}^{M} v_j \tag{C.28}$$

Proof. Assume that service j takes configuration $\boldsymbol{\lambda}_0$ after its placement (and before it possibly unpredictably departs), then $\mathbf{y}_j^{(j)} - \mathbf{y}_{j-1}^{(j)} = \mathbf{a}_{j\boldsymbol{\lambda}_0}$ and $\mathbf{z}_j^{(j)} - \mathbf{z}_{j-1}^{(j)} = \mathbf{b}_{j\boldsymbol{\lambda}_0}$. For any time after placing instance j we define $\Delta\mathbf{y}_j \triangleq \mathbf{y}_j - \mathbf{y}_{j-1}$ and $\Delta\mathbf{z}_j \triangleq \mathbf{z}_j - \mathbf{z}_{j-1}$. We always have $\Delta\mathbf{y}_j = \mathbf{a}_{j\boldsymbol{\lambda}_0}$, $\Delta\mathbf{z}_j = \mathbf{b}_{j\boldsymbol{\lambda}_0}$, if instance j has not yet departed from the system, and $\Delta\mathbf{y}_j = \Delta\mathbf{z}_j = 0$ if j has already departed from the system.

Noting that $\widetilde{D}\left(\mathbf{y}_j, \mathbf{z}_j\right)$ is convex non-decreasing (from Lemma 5.1), we have

$$\begin{aligned}
&\widetilde{D}\left(\mathbf{y}_j, \mathbf{z}_j\right) - \widetilde{D}\left(\mathbf{y}_{j-1}, \mathbf{z}_{j-1}\right) \\
&= \widetilde{D}\left(\mathbf{y}_{j-1} + \Delta\mathbf{y}_j, \mathbf{z}_{j-1} + \Delta\mathbf{z}_j\right) - \widetilde{D}\left(\mathbf{y}_{j-1}, \mathbf{z}_{j-1}\right) \\
&\leq \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)} + \Delta\mathbf{y}_j, \mathbf{z}_{j-1}^{(j)} + \Delta\mathbf{z}_j\right) - \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}, \mathbf{z}_{j-1}^{(j)}\right) && \text{(C.29)} \\
&\leq \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)} + \mathbf{a}_{j\boldsymbol{\lambda}_0}, \mathbf{z}_{j-1}^{(j)} + \mathbf{b}_{j\boldsymbol{\lambda}_0}\right) - \widetilde{D}\left(\mathbf{y}_{j-1}^{0}, \mathbf{z}_{j-1}^{0}\right) && \text{(C.30)} \\
&= \widetilde{D}\left(\mathbf{y}_j^{(j)}, \mathbf{z}_j^{(j)}\right) - \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}, \mathbf{z}_{j-1}^{(j)}\right) \\
&= v_j
\end{aligned}$$

where inequality (C.29) is because $\mathbf{y}_{j-1} \leq \mathbf{y}_{j-1}^{(j)}$, $\mathbf{z}_{j-1} \leq \mathbf{z}_{j-1}^{(j)}$ (see discussion above) and due to the convex non-decreasing property of $\widetilde{D}\left(\mathbf{y}_j, \mathbf{z}_j\right)$; inequality (C.30) is because $\Delta\mathbf{y}_j \leq \mathbf{a}_{j\boldsymbol{\lambda}_0}$, $\Delta\mathbf{z}_j \leq \mathbf{b}_{j\boldsymbol{\lambda}_0}$ and also due to the non-decreasing property of $\widetilde{D}\left(\mathbf{y}_j, \mathbf{z}_j\right)$.

We now note that $\widetilde{D}\left(\mathbf{x}_0\right) = 0$, where $\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{0}$ is defined as a vector with all zeros, thus $\mathbf{y}_0 = \mathbf{z}_0 = \mathbf{0}$. We have

$$\begin{aligned}\sum_{j=1}^{M} v_j &\geq \sum_{j=1}^{M} \left[\widetilde{D}\left(\mathbf{y}_j, \mathbf{z}_j\right) - \widetilde{D}\left(\mathbf{y}_{j-1}, \mathbf{z}_{j-1}\right)\right] \\ &= \widetilde{D}\left(\mathbf{x}_M\right) - \widetilde{D}\left(\mathbf{x}_0\right) = \widetilde{D}\left(\mathbf{x}_M\right)\end{aligned}$$

□

Lemma C.4. *When Assumption 5.1 is satisfied, for any j and $\boldsymbol{\lambda}$, we have*

$$v_j \leq \phi \frac{\partial \widetilde{D}}{\partial x_{j\boldsymbol{\lambda}}}\left(\mathbf{x}_M\right) \tag{C.31}$$

where ϕ is a constant satisfying (5.12).

Proof. Assume that service j takes configuration $\boldsymbol{\lambda}_0$ after its placement (and before it possibly unpredictably departs). Because we perform a greedy assignment in Algorithm 5.3, we have

$$\begin{aligned}v_j &= \widetilde{D}\left(\mathbf{y}_j^{(j)}, \mathbf{z}_j^{(j)}\right) - \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}, \mathbf{z}_{j-1}^{(j)}\right) \\ &= \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)} + \mathbf{a}_{j\boldsymbol{\lambda}_0}, \mathbf{z}_{j-1}^{(j)} + \mathbf{b}_{j\boldsymbol{\lambda}_0}\right) - \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}, \mathbf{z}_{j-1}^{(j)}\right) \\ &\leq \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)} + \mathbf{a}_{j\boldsymbol{\lambda}}, \mathbf{z}_{j-1}^{(j)} + \mathbf{b}_{j\boldsymbol{\lambda}}\right) - \widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}, \mathbf{z}_{j-1}^{(j)}\right)\end{aligned}$$

for any $\boldsymbol{\lambda} \in \Lambda_i$.

Then, we have

$$\begin{aligned}
&\widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}+\mathbf{a}_{j\boldsymbol{\lambda}},\mathbf{z}_{j-1}^{(j)}+\mathbf{b}_{j\boldsymbol{\lambda}}\right)-\widetilde{D}\left(\mathbf{y}_{j-1}^{(j)},\mathbf{z}_{j-1}^{(j)}\right)\\
&\leq\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left(\mathbf{y}_{j-1}^{(j)}+\mathbf{a}_{j\boldsymbol{\lambda}},\mathbf{z}_{j-1}^{(j)}+\mathbf{b}_{j\boldsymbol{\lambda}}\right)\cdot(\mathbf{a}_{j\boldsymbol{\lambda}},\mathbf{b}_{j\boldsymbol{\lambda}}) &\text{(C.32)}\\
&\leq\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left(\mathbf{y}_{\max}+\mathbf{a}_{j\boldsymbol{\lambda}},\mathbf{z}_{\max}+\mathbf{b}_{j\boldsymbol{\lambda}}\right)\cdot(\mathbf{a}_{j\lambda},\mathbf{b}_{j\boldsymbol{\lambda}}) &\text{(C.33)}\\
&\leq\phi\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left(\mathbf{y}_{M},\mathbf{z}_{M}\right)\cdot(\mathbf{a}_{j\boldsymbol{\lambda}},\mathbf{b}_{j\boldsymbol{\lambda}}) &\text{(C.34)}\\
&=\phi\frac{\partial\widetilde{D}}{\partial x_{j\boldsymbol{\lambda}}}\left(\mathbf{x}_{M}\right) &\text{(C.35)}
\end{aligned}$$

where "$\cdot$" denotes the dot-product. The above relationship is explained as follows. Inequality (C.32) follows from the first-order conditions of convex functions [56, Section 3.1.3]. The definition of $\mathbf{y}_{\max}$ and $\mathbf{z}_{\max}$ in Proposition 5.2 gives (C.33). The definition of ϕ in (5.12) gives (C.34). Equality (C.35) follows from Lemma C.1. This completes the proof. □

Using the above lemmas, we now proof Proposition 5.2.

Proof. **(Proposition 5.2)** Due to the convexity of $\widetilde{D}(\mathbf{x})$, from the first-order conditions of convex functions [56, Section 3.1.3], we have

$$\begin{aligned}
&\widetilde{D}(\phi\psi\mathbf{x}_{M}^{*})-\widetilde{D}(\mathbf{x}_{M})\\
&\geq\nabla_{\mathbf{x}}\widetilde{D}\left(\mathbf{x}_{M}\right)\cdot(\phi\psi\mathbf{x}_{M}^{*}-\mathbf{x}_{M}) &\text{(C.36)}\\
&=\phi\psi\nabla_{\mathbf{x}}\widetilde{D}\left(\mathbf{x}_{M}\right)\cdot\mathbf{x}_{M}^{*}-\nabla_{\mathbf{x}}\widetilde{D}\left(\mathbf{x}_{M}\right)\mathbf{x}_{M} &\text{(C.37)}\\
&=\sum_{i=1}^{M}\sum_{\boldsymbol{\lambda}\in\Lambda}\phi\psi x_{i\boldsymbol{\lambda}}^{*}\frac{\partial\widetilde{D}}{\partial x_{i\boldsymbol{\lambda}}}\left(\mathbf{x}_{M}\right)-\nabla_{\mathbf{x}}\widetilde{D}\left(\mathbf{x}_{M}\right)\cdot\mathbf{x}_{M} &\text{(C.38)}\\
&=\psi\left(\sum_{i=1}^{M}\sum_{\boldsymbol{\lambda}\in\Lambda}x_{i\boldsymbol{\lambda}}^{*}\phi\frac{\partial\widetilde{D}}{\partial x_{i\boldsymbol{\lambda}}}\left(\mathbf{x}_{M}\right)-\frac{\nabla_{\mathbf{x}}\widetilde{D}\left(\mathbf{x}_{M}\right)\cdot\mathbf{x}_{M}}{\psi}\right) &\text{(C.39)}
\end{aligned}$$

where $x_{i\boldsymbol{\lambda}}^{*}$ is the $(i,\boldsymbol{\lambda})$th element of vector $\mathbf{x}_{M}^{*}$. From Lemma C.4, we have

$$\text{Eq. (C.39)} \geq \psi \left(\sum_{i=1}^{M} \sum_{\boldsymbol{\lambda} \in \Lambda} x^*_{i\boldsymbol{\lambda}} v_i - \frac{\nabla_{\mathbf{x}} \widetilde{D}\left(\mathbf{x}_M\right) \cdot \mathbf{x}_M}{\psi} \right) \tag{C.40}$$

$$= \psi \left(\sum_{i=1}^{M} v_i \sum_{\boldsymbol{\lambda} \in \Lambda} x^*_{i\boldsymbol{\lambda}} - \frac{\nabla_{\mathbf{x}} \widetilde{D}\left(\mathbf{x}_M\right) \cdot \mathbf{x}_M}{\psi} \right) \tag{C.41}$$

From the constraint $\sum_{\boldsymbol{\lambda} \in \Lambda} x^*_{i\boldsymbol{\lambda}} = 1$ and the definition of ψ, we get

$$\text{Eq. (C.41)} = \psi \left(\sum_{i=1}^{M} v_i - \frac{\nabla_{\mathbf{x}} \widetilde{D}\left(\mathbf{x}_M\right) \cdot \mathbf{x}_M}{\psi} \right) \tag{C.42}$$

$$\geq \psi \left(\sum_{i=1}^{M} v_i - \widetilde{D}(\mathbf{x}_M) \right) \tag{C.43}$$

$$\geq 0 \tag{C.44}$$

where the last equality follows from Lemma C.3. This gives (5.10).

Equation (5.11) follows from the fact that $y_{n,j}(t)$ and $z_{nh,j}(t)$ are both linear in $x_{i\lambda}$.

The last equality in (5.13) follows from Lemma C.2 and the fact that $\widetilde{D}\left(\mathbf{x}\right) = \widetilde{D}\left(\mathbf{y}, \mathbf{z}\right)$ as well as $\mathbf{x} = \mathbf{x}_M$, $\mathbf{y} = \mathbf{y}_M$, and $\mathbf{z} = \mathbf{z}_M$. □

C.5 Proof of Proposition 5.4

Lemma C.5. *For polynomial functions $\Xi_1(y)$ and $\Xi_2(y)$ in the general form:*

$$\Xi_1(y) \triangleq \sum_{\rho=0}^{\Omega} \omega_1^{(\rho)} y^\rho$$
$$\Xi_2(y) \triangleq \sum_{\rho=0}^{\Omega} \omega_2^{(\rho)} y^\rho$$

where the constants $\omega_1^{(\rho)} \geq 0$ and $\omega_2^{(\rho)} \geq 0$ for $0 \leq \rho < \Omega$, while $\omega_1^{(\Omega)} > 0$ and $\omega_2^{(\Omega)} > 0$, we have

$$\lim_{y \to +\infty} \frac{\Xi_1(y)}{\Xi_2(y)} = \frac{\omega_1^{(\Omega)}}{\omega_2^{(\Omega)}}$$

Proof. When $\Omega = 0$, we have

$$\lim_{y \to +\infty} \frac{\Xi_1(y)}{\Xi_2(y)} = \frac{\omega_1^{(0)}}{\omega_2^{(0)}}$$

When $\Omega > 0$, we note that $\lim_{y \to +\infty} \Xi_1(y) = +\infty$ and $\lim_{y \to +\infty} \Xi_2(y) = +\infty$, because $\omega_1^{(\Omega)} > 0$ and $\omega_2^{(\Omega)} > 0$. We apply the L'Hospital's rule and get

$$\begin{aligned} \lim_{y \to +\infty} \frac{\Xi_1(y)}{\Xi_2(y)} &= \lim_{y \to +\infty} \frac{\frac{d\Xi_1(y)}{dy}}{\frac{d\Xi_2(y)}{dy}} \\ &= \lim_{y \to +\infty} \frac{\sum_{\rho=1}^{\Omega} \rho \omega_1^{(\rho)} y^{\rho-1}}{\sum_{\rho=1}^{\Omega} \rho \omega_2^{(\rho)} y^{\rho-1}} \end{aligned} \tag{C.45}$$

Suppose we have

$$\lim_{y \to +\infty} \frac{\Xi_1(y)}{\Xi_2(y)} = \lim_{y \to +\infty} \frac{\sum_{\rho=k}^{\Omega} \left(\prod_{m=0}^{k-1} (\rho - m) \right) \omega_1^{(\rho)} y^{\rho-k}}{\sum_{\rho=k}^{\Omega} \left(\prod_{m=0}^{k-1} (\rho - m) \right) \omega_2^{(\rho)} y^{\rho-k}} \tag{C.46}$$

which equals to (C.45) for $k = 1$. For $1 \leq k < \Omega$, we note that $\Omega - k > 0$, hence the numerator and denominator in the right hand-side (RHS) of (C.46) still respectively

approach $+\infty$ when $y \to +\infty$ (because $\omega_1^{(\Omega)} > 0$ and $\omega_2^{(\Omega)} > 0$). Let $\Psi(k)$ denote the RHS (C.46), we can reapply the L'Hospital's rule on $\Psi(k)$, yielding

$$\begin{aligned}\Psi(k) &= \lim_{y\to+\infty} \frac{\sum_{\rho=k+1}^{\Omega} \left(\prod_{m=0}^{(k+1)-1}(\rho-m)\right) \omega_1^{(\rho)} y^{\rho-(k+1)}}{\sum_{\rho=k+1}^{\Omega} \left(\prod_{m=0}^{(k+1)-1}(\rho-m)\right) \omega_2^{(\rho)} y^{\rho-(k+1)}} \\ &= \Psi(k+1)\end{aligned}$$

which proofs that (C.46) holds for $1 \leq k \leq \Omega$. Therefore,

$$\lim_{y\to+\infty} \frac{\Xi_1(y)}{\Xi_2(y)} = \Psi(\Omega) = \frac{\rho!\omega_1^{(\Omega)}}{\rho!\omega_2^{(\Omega)}} = \frac{\omega_1^{(\Omega)}}{\omega_2^{(\Omega)}}$$

□

Lemma C.6. *For variables* $0 \leq y \leq y', 0 \leq y_n \leq y'_n, 0 \leq y_h \leq y'_h, 0 \leq z_{nh} \leq z'_{nh}$, *we always have*

$$\frac{du_{n,t}}{dy}(y) \leq \frac{du_{n,t}}{dy}(y') \tag{C.47}$$

$$\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n, y_h, z_{nh}) \leq \frac{\partial w_{nh,t}}{\partial \Upsilon}(y'_n, y'_h, z'_{nh}) \tag{C.48}$$

where Υ *stands for either* y_n, y_h, *or* z_{nh}.

Proof. We note that

$$\frac{du_{n,t}}{dy}(y) = \sum_{\rho} \rho \gamma_{n,t}^{(\rho)} y^{\rho-1}$$

from which (C.47) follows directly because $\gamma_{n,t}^{(\rho)} \geq 0$. We then note that

$$\frac{\partial w_{nh,t}}{\partial y_n}(y_n, y_h, z_{nh}) = \sum_{\rho_1}\sum_{\rho_2}\sum_{\rho_3} \rho_1 \kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} y_n^{\rho_1-1} y_h^{\rho_2} z_{nh}^{\rho_3}$$

from which (C.48) follows for $\Upsilon = y_n$ because $\kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} \geq 0$. Similarly, (C.48) also follows for $\Upsilon = y_h$ and $\Upsilon = z_{nh}$. □

Lemma C.7. *Let Ω denote the highest order of the polynomial cost functions. Specifically, $\Omega \triangleq \max\{\rho; \rho_1 + \rho_2 + \rho_3\}$, subject to $\gamma_{n,t}^{(\rho)} > 0$ and $\kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} > 0$. Assume that the cost functions are defined as in (5.16) and (5.17), then for any constants $\delta > 0$, $B \geq 0$, there exist sufficiently large values of y, y_n, y_h, z_{nh}, such that*

$$\frac{\frac{du_{n,t}}{dy}(y+B)}{\frac{du_{n,t}}{dy}(y)} \leq 1+\delta \tag{C.49}$$

$$\frac{\frac{du_{n,t}}{dy}(y)\cdot y}{u_{n,t}(y)} \leq \Omega+\delta \tag{C.50}$$

$$\frac{\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n+B, y_h+B, z_{nh}+B)}{\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n, y_h, z_{nh})} \leq 1+\delta \tag{C.51}$$

$$\frac{\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n, y_h, z_{nh})\cdot \Upsilon}{w_{n,t}(y_n, y_h, z_{nh})} \leq \Omega+\delta \tag{C.52}$$

for any n, h, t, where Υ stands for either y_n, y_h, or z_{nh}.

Proof. Let Ω' denote the maximum value of ρ such that $\gamma_{n,t}^{(\rho)} > 0$, we always have $\Omega' \leq \Omega$. We note that

$$\frac{\frac{du_{n,t}}{dy}(y+B)}{\frac{du_{n,t}}{dy}(y)} = \frac{\sum_{\rho=1}^{\Omega'} \rho\gamma_{n,t}^{(\rho)}(y+B)^{\rho-1}}{\sum_{\rho=1}^{\Omega'} \rho\gamma_{n,t}^{(\rho)} y^{\rho-1}} \tag{C.53}$$

$$\frac{\frac{du_{n,t}}{dy}(y)\cdot y}{u_{n,t}(y)} = \frac{\sum_{\rho=1}^{\Omega'} \rho\gamma_{n,t}^{(\rho)} y^{\rho}}{\sum_{\rho=1}^{\Omega'} \gamma_{n,t}^{(\rho)} y^{\rho}} \tag{C.54}$$

According to Lemma C.5, we have

$$\lim_{y\to+\infty} \frac{\frac{du_{n,t}}{dy}(y+B)}{\frac{du_{n,t}}{dy}(y)} = \frac{\Omega'\gamma_{n,t}^{(\Omega')}}{\Omega'\gamma_{n,t}^{(\Omega')}} = 1 \tag{C.55}$$

$$\lim_{y\to+\infty} \frac{\frac{du_{n,t}}{dy}(y)\cdot y}{u_{n,t}(y)} = \frac{\Omega'\gamma_{n,t}^{(\Omega')}}{\gamma_{n,t}^{(\Omega')}} = \Omega' \tag{C.56}$$

where we note that after expanding the numerator in the RHS of (C.53), the constant B does not appear in the coefficient of $y^{\Omega'-1}$.

Now, define a variable $q > 0$, and we let $y_n = \zeta_1 q, y_h = \zeta_2 q, z_{nh} = \zeta_3 q$, where $\zeta_1, \zeta_2, \zeta_3 > 0$ are arbitrary constants. We also define $\rho \triangleq \rho_1+\rho_2+\rho_3$. Using $\zeta_1, \zeta_2, \zeta_3$, and q, we can represent any value of $(y_n, y_h, z_{nh}) > \mathbf{0}$. With this definition, we have

$$\begin{aligned} w_{nh,t}(q) &\triangleq w_{nh,t}(\zeta_1 q, \zeta_2 q, \zeta_3 q) \\ &= \sum_{\rho_1}\sum_{\rho_2}\sum_{\rho_3} \kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} \zeta_1^{\rho_1}\zeta_2^{\rho_2}\zeta_3^{\rho_3} q^{\rho_1+\rho_2+\rho_3} \\ &= \sum_{\rho=1}^{\Omega''} (\kappa')_{nh,t}^{(\rho)} q^{\rho} \end{aligned} \tag{C.57}$$

where the constant

$$(\kappa')_{nh,t}^{(\rho)} \triangleq \sum_{\{(\rho_1,\rho_2,\rho_3):\rho_1+\rho_2+\rho_3=\rho\}} \kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} \zeta_1^{\rho_1}\zeta_2^{\rho_2}\zeta_3^{\rho_3}$$

and Ω'' is defined as the maximum value of ρ such that $(\kappa')_{nh,t}^{(\rho)} > 0$. We always have $\Omega'' \leq \Omega$. Note that (C.57) is in the same form as (5.16). Following the same procedure as for obtaining (C.55) and (C.56), we get

$$\lim_{q\to+\infty} \frac{\frac{dw_{nh,t}}{dq}(q+B')}{\frac{dw_{nh,t}}{dq}(q)} = \frac{\Omega''\gamma_{n,t}^{(\Omega'')}}{\Omega''\gamma_{n,t}^{(\Omega'')}} = 1 \tag{C.58}$$

$$\lim_{q\to+\infty} \frac{\frac{dw_{nh,t}}{dq}(q)\cdot q}{w_{n,t}(q)} = \frac{\Omega''\gamma_{n,t}^{(\Omega'')}}{\gamma_{n,t}^{(\Omega'')}} = \Omega'' \tag{C.59}$$

where $B' \triangleq \frac{B}{\min\{\zeta_1;\zeta_2;\zeta_3\}}$.

According to the definition of limits, for any $\delta > 0$, there exist sufficiently large values of y and q (thus y_n, y_h, z_{nh}), such that

$$\frac{\frac{du_{n,t}}{dy}(y+B)}{\frac{du_{n,t}}{dy}(y)} \leq 1+\delta \tag{C.60}$$

$$\frac{\frac{du_{n,t}}{dy}(y)\cdot y}{u_{n,t}(y)} \leq \Omega'+\delta \tag{C.61}$$

$$\frac{\frac{dw_{nh,t}}{dq}(q+B')}{\frac{dw_{nh,t}}{dq}(q)} \leq 1+\delta \tag{C.62}$$

$$\frac{\frac{dw_{nh,t}}{dq}(q)\cdot q}{w_{n,t}(q)} \leq \Omega''+\delta \tag{C.63}$$

for any n, h, t.

Because

$$\frac{\frac{dw_{nh,t}}{dq}(q+B')}{\frac{dw_{nh,t}}{dq}(q)} = \frac{\frac{dw_{nh,t}}{d(\zeta q)}(q+B')}{\frac{dw_{nh,t}}{d(\zeta q)}(q)}$$

$$\frac{dw_{nh,t}}{dq}(q)\cdot q = \frac{dw_{nh,t}}{d(\zeta q)}(q)\cdot \zeta q$$

for any $\zeta > 0$, we can also express the bounds (C.62) and (C.63) in terms of y_n, y_h, z_{nh}, yielding

$$\frac{\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n+\zeta_1 B', y_h+\zeta_2 B', z_{nh}+\zeta_3 B')}{\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n, y_h, z_{nh})} \leq 1+\delta \tag{C.64}$$

$$\frac{\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n, y_h, z_{nh})\cdot \Upsilon}{w_{n,t}(y_n, y_h, z_{nh})} \leq \Omega''+\delta \tag{C.65}$$

where Υ stands for either y_n, y_h, or z_{nh}. According to the definition of B', we have $B \leq \zeta_1 B'$, $B \leq \zeta_2 B'$, $B \leq \zeta_3 B'$. From Lemma C.6, we have

$$\begin{aligned}&\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n+B, y_h+B, z_{nh}+B)\\ &\leq \frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n+\zeta_1 B', y_h+\zeta_2 B', z_{nh}+\zeta_3 B')\end{aligned} \tag{C.66}$$

Combining (C.66) with (C.64) and noting that $\Omega' \leq \Omega$ and $\Omega'' \leq \Omega$, together with (C.60), (C.61), and (C.65), we get (C.49)–(C.52). □

Lemma C.8. *For arbitrary values $\vartheta_{1,k} \geq 0$ and $\vartheta_{2,k} \geq 0$ for all $k = 1, 2, ..., K$, where $\vartheta_{1,k}$ and $\vartheta_{2,k}$ are either both zero or both non-zero and there exists k such that*

$\vartheta_{1,k}$ and $\vartheta_{2,k}$ are non-zero, if the following bound is satisfied:

$$\max_{\{k\in\{1,\ldots,K\}:\vartheta_{1,k}\neq 0,\vartheta_{2,k}\neq 0\}} \frac{\vartheta_{1,k}}{\vartheta_{2,k}} \leq \Theta$$

then we have

$$\frac{\sum_{k=1}^{K}\omega_k\vartheta_{1,k}}{\sum_{k=1}^{K}\omega_k\vartheta_{2,k}} \leq \Theta$$

for any $\omega_k \geq 0$.

Proof. Because $\vartheta_{1,k} \leq \Theta\vartheta_{2,k}$ for all k, we have

$$\sum_{k=1}^{K}\omega_k\vartheta_{1,k} \leq \sum_{k=1}^{K}\omega_k\Theta\vartheta_{2,k}$$

yielding the result. □

Lemma C.9. *When Assumption 5.2 is satisfied and the window size T is a constant, there exists a constant $B \geq 0$ such that*

$$(\mathbf{y}_{max} + \mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{z}_{max} + \mathbf{b}_{i\boldsymbol{\lambda}}) - (\mathbf{y}, \mathbf{z}) \leq B\mathbf{e} \tag{C.67}$$

for any i and any $\boldsymbol{\lambda} \in \Lambda_i$, where $\mathbf{e} \triangleq [1, ..., 1]$ is a vector of all ones that has the same dimension as $(\mathbf{y}, \mathbf{z})$.

Proof. We note that

$$\begin{aligned}
&(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{z}_{\max} + \mathbf{b}_{i\boldsymbol{\lambda}}) - (\mathbf{y}, \mathbf{z}) \\
&\leq (\mathbf{y}_{\max} + a_{\max}\mathbf{e}_{\mathbf{y}}, \mathbf{z}_{\max} + b_{\max}\mathbf{e}_{\mathbf{z}}) - (\mathbf{y}, \mathbf{z}) && \text{(C.68)} \\
&\leq (a_{\max}(B_dT + 1)\,\mathbf{e}_{\mathbf{y}}, b_{\max}(B_dT + 1)\,\mathbf{e}_{\mathbf{z}}) && \text{(C.69)} \\
&\leq \max\{a_{\max}(B_dT + 1)\,; b_{\max}(B_dT + 1)\} \cdot \mathbf{e} && \text{(C.70)}
\end{aligned}$$

where $\mathbf{e}_{\mathbf{y}} \triangleq [1, ..., 1]$ and $\mathbf{e}_{\mathbf{z}} \triangleq [1, ..., 1]$ are vectors of all ones that respectively have

the same dimensions as $\mathbf{y}$ and $\mathbf{z}$. Inequality (C.68) follows from the boundedness assumption in Assumption 5.2. Inequality (C.69) follows by noting that the gap between $(\mathbf{y}_{\max}, \mathbf{z}_{\max})$ and $(\mathbf{y}, \mathbf{z})$ is because of instances unpredictably leaving the system before their maximum lifetime, and that there are at most T slots, at most B_d instances unpredictably leave the system in each slot (according to Assumption 5.2). Inequality (C.70) is obvious (note that the maximum is taken element-wise).

By setting $B = \max\left\{a_{\max}\left(B_d T + 1\right); b_{\max}\left(B_d T + 1\right)\right\}$, we prove the result. □

We now proof Proposition 5.4.

Proof. **(Proposition 5.4)** We note that $\widetilde{D}\left(\mathbf{y}, \mathbf{z}\right)$ sums up $u_{n,t}(y_n)$ and $w_{nh,t}\left(y_n, y_h, z_{nh}\right)$ over t, n, h, as defined in (5.7).

The numerator in the RHS of (5.12) can be expanded into a sum containing terms of either

$$\frac{du_{n,t}}{dy}\left(\left(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}\right)_{nt}\right)$$

or

$$\frac{\partial w_{nh,t}}{\partial \Upsilon}\left(\left(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}\right)_{nt}, \left(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}\right)_{ht}, \left(\mathbf{z}_{\max} + \mathbf{b}_{i\boldsymbol{\lambda}}\right)_{nht}\right)$$

where Υ stands for either $y_n(t)$, $y_h(t)$, or $z_{nh}(t)$, with either $a_{i\boldsymbol{\lambda}n}(t)$ or $b_{i\boldsymbol{\lambda}nh}(t)$ as weights. Because Assumption 5.2 is satisfied, according to (C.67) in Lemma C.9, there exists a constant $B \geq 0$ such that

$$\left(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}\right)_{nt} \leq y_n(t) + B$$

$$\left(\mathbf{z}_{\max} + \mathbf{b}_{i\boldsymbol{\lambda}}\right)_{nht} \leq z_{nh}(t) + B$$

for all n, h, t. From Lemma C.6, we have

$$\frac{du_{n,t}}{dy}\left(y_n(t)+B\right) \geq \frac{du_{n,t}}{dy}\left(\left(\mathbf{y}_{\max}+\mathbf{a}_{i\boldsymbol{\lambda}}\right)_{nt}\right)$$

and

$$\begin{aligned}&\frac{\partial w_{nh,t}}{\partial \Upsilon}\left(y_n(t)+B, y_h(t)+B, z_{nh}(t)+B\right)\\ &\geq \frac{\partial w_{nh,t}}{\partial \Upsilon}\left(\left(\mathbf{y}_{\max}+\mathbf{a}_{i\boldsymbol{\lambda}}\right)_{nt}, \left(\mathbf{y}_{\max}+\mathbf{a}_{i\boldsymbol{\lambda}}\right)_{ht}, \left(\mathbf{z}_{\max}+\mathbf{b}_{i\boldsymbol{\lambda}}\right)_{nht}\right)\end{aligned}$$

Therefore, if

$$\phi \geq \frac{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left((\mathbf{y},\mathbf{z})+B\mathbf{e}\right)\cdot\left(\mathbf{a}_{i\boldsymbol{\lambda}},\mathbf{b}_{i\boldsymbol{\lambda}}\right)}{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\left(\mathbf{y},\mathbf{z}\right)\cdot\left(\mathbf{a}_{i\boldsymbol{\lambda}},\mathbf{b}_{i\boldsymbol{\lambda}}\right)} \tag{C.71}$$

then (5.12) is always satisfied. Similar to the above, the numerator in the RHS of (C.71) can be expanded into a sum containing terms of either $\frac{du_{n,t}}{dy}\left(y_n(t)+B\right)$ and $\frac{\partial w_{nh,t}}{\partial \Upsilon}\left(y_n(t)+B, y_h(t)+B, z_{nh}(t)+B\right)$ with either $a_{i\boldsymbol{\lambda}n}(t)$ or $b_{i\boldsymbol{\lambda}nh}(t)$ as weights.

Again, the denominator in the RHS of (5.12) (or equivalently, (C.71)) can be expanded into a sum containing terms of either $\frac{du_{n,t}}{dy}(y(t))$ or $\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n(t), y_h(t), z_{nh}(t))$, with either $a_{i\boldsymbol{\lambda}n}(t)$ or $b_{i\boldsymbol{\lambda}nh}(t)$ as weights.

For any given $i, \boldsymbol{\lambda}$, the terms $\frac{du_{n,t}}{dy}(y_n(t)+B)$ and $\frac{du_{n,t}}{dy}(y_n(t))$ have the same weight $a_{i\boldsymbol{\lambda}n}(t)$, and $\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n(t)+B, y_h(t)+B, z_{nh}(t)+B)$ and $\frac{\partial w_{nh,t}}{\partial \Upsilon}(y_n(t), y_h(t), z_{nh}(t))$ have the same weight $b_{i\boldsymbol{\lambda}nh}(t)$. According to Lemmas C.7 and C.8, for any $\delta > 0$, there exist sufficiently large values of $\mathbf{y}$ and $\mathbf{z}$, such that

$$\text{RHS of (C.71)} \leq 1+\delta$$

Following a similar reasoning, we know that, for any $\delta > 0$, there exist sufficiently large values of $\mathbf{y}$ and $\mathbf{z}$, such that

$$\text{RHS of (5.13)} \leq \Omega+\delta$$

where we recall that Ω denotes the highest order of the polynomial cost functions, i.e., $\Omega \triangleq \max\{\rho; \rho_1 + \rho_2 + \rho_3\}$, subject to $\gamma_{n,t}^{(\rho)} > 0$ and $\kappa_{nh,t}^{(\rho_1,\rho_2,\rho_3)} > 0$.

We assume sufficiently large $\mathbf{y}, \mathbf{z}$ in the following, in which case we can set $\phi = 1 + \delta$ and $\psi = \Omega + \delta$ while satisfying (C.71) (thus (5.12)) and (5.13).

We then note that from (5.16), (5.17), and the definition of Ω, we have

$$\begin{aligned}\widetilde{D}(\phi\psi\mathbf{x}^*) &\leq (\phi\psi)^{\Omega}\widetilde{D}(\mathbf{x}^*) \\ &= ((1+\delta)(\Omega+\delta))^{\Omega}\widetilde{D}(\mathbf{x}^*) \\ &= \left(\Omega^{\Omega} + \delta'\right)\widetilde{D}(\mathbf{x}^*)\end{aligned}$$

where $\delta' \triangleq \delta\Omega + \delta + \delta^2 > 0$ is an arbitrary constant (because δ is an arbitrary constant). The first inequality is because of $\phi, \psi \geq 1$ and $\widetilde{D}(\phi\psi\mathbf{x}^*)$ is a polynomial of $\phi\psi\mathbf{x}^*$ with maximum order of Ω, where we note that $\mathbf{y}$ and $\mathbf{z}$ are both linear in $\mathbf{x}$.

We then have

$$\frac{\widetilde{D}(\mathbf{x})}{\widetilde{D}(\mathbf{x}^*)} \leq \frac{\widetilde{D}(\phi\psi\mathbf{x}^*)}{\widetilde{D}(\mathbf{x}^*)} = \Omega^{\Omega} + \delta' \tag{C.72}$$

Until now, we have shown that (C.72) holds for sufficiently large $\mathbf{y}$ and $\mathbf{z}$. According to Assumption 5.2, the number of instances that unpredictably leave the system in each slot is upper bounded by a constant B_d. It follows that $\mathbf{y}$ and $\mathbf{z}$ increase with M when M is larger than a certain threshold. Therefore, there exists a sufficiently large M, so that we have a sufficiently large $\mathbf{y}$ and $\mathbf{z}$ that satisfies (C.72).

Hence, the competitive ratio upper bound can be expressed as

$$\Gamma \triangleq \max_{\mathcal{I}(M)} \Gamma(\mathcal{I}(M)) \leq \Omega^{\Omega} + \delta' \tag{C.73}$$

for sufficiently large M.

According to the definition of the big-O notation, we can also write

$$\Gamma = O(1) \tag{C.74}$$

because Ω and δ' are both constants in M. □