Data Caching in Edge Computing

By

Xiaoyu Xia

Submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Deakin University

August 2021

To my family...

Contents

Acknowledgements

My deepest and sincere gratitude goes first and foremost to my respected supervisors, Dr. Feifei Chen, A/Prof. Qiang He, Prof. John Grundy and A/Prof. Mohamed Abdelrazek, for their constant encouragement, guidance, devotion and constructive feedback. They guided me through difficulties when I struggled and encouraged me when I feel depressed. I appreciate all the time they have spent on discussing my research ideas, listening to my problems, editing my papers, and taking away all my doubts and worries.

I would like to give special thanks to Feifei and Qiang for giving me the opportunity to study at Deakin University. I still remember their warm replies when I contacted them at the PhD application stage. I am also greatly indebted to John for his valuable career advises and continuing supports.

I also owe a great deal of thanks to my research friends who helped me during my research and daily life during my PhD study: Guangming Cui, Bo Li, Phu Lai, Jingwen Zhou, Liang Yuan, Bowen Liu, Ruikun Luo and many others.

I wish to express my heartfelt gratitude to my wife, Baobao Pan, who has stood by me through all my happy moments, my travails, my absences, my fits of pique and impatience. Without her continuous support and help in both study and life, I cannot make this.

Last but not least my thanks go to my beloved parents and parents-in-law for their loving considerations and great confidence in me all through these years.

Melbourne, Australia
Xiaoyu Xia
August, 2021

Publication List

The papers completed in this PhD study are listed below, and the authorship statements for the papers included in this thesis are attached after the publications.

First-author Papers

1. **Xiaoyu Xia**, Feifei Chen, Qiang He, Guangming Cui, John Grundy, Mohamed Abdelrazek, Athman Bouguettaya, Hai Jin, OL-MEDC: An Online Approach for Cost-effective Data Caching in Mobile Edge Computing Systems, *IEEE Transactions on Mobile Computing* (**TMC, CORE A***), accepted in 2021. DOI:10.1109/TMC.2021.3107918.

2. **Xiaoyu Xia**, Feifei Chen, Qiang He, Guangming Cui, John Grundy, Mohamed Abdelrazek, Xiaolong Xu, Hai Jin, Data, user and power allocations for caching in multi-access edge computing, *IEEE Transactions on Parallel and Distributed Systems* (**TPDS, CORE A***), accepted in 2021. DOI:10.1109/TPDS.2021.3104241.

3. **Xiaoyu Xia**, Feifei Chen, John Grundy, Mohamed Abdelrazek, Hai Jin, Qiang He, Constrained App Data Caching over Edge Server Graphs in Edge Computing Environment, *IEEE Transactions on Services Computing* (**TSC, CORE A***), accepted in 2021. DOI:10.1109/TSC.2021.3062017.

4. **Xiaoyu Xia**, Feifei Chen, Qiang He, John Grundy, Mohamed Abdelrazek, Hai Jin, Online Collaborative Data Caching in Edge Computing, *IEEE Transactions*

on Parallel and Distributed Systems (**TPDS, CORE A***), Vol. 32(2), pp. 281-294, 2020.

5. **Xiaoyu Xia**, Feifei Chen, Qiang He, John Grundy, Mohamed Abdelrazek, Hai Jin, Cost-Effective App Data Distribution in Edge Computing, *IEEE Transactions on Parallel and Distributed Systems* (**TPDS, CORE A***), Vol. 32(1), pp. 31-44, 2020.

6. **Xiaoyu Xia**, Feifei Chen, Qiang He, Guangming Cui, Phu Lai, Mohamed Abdelrazek, John Grundy, Hai Jin, Graph-based Data Caching Optimization in Edge Computing, *Future Generation Computer Systems* (**FGCS , CORE A**), Vol. 112, pp. 684-694, 2020.

7. **Xiaoyu Xia**, Feifei Chen, Guangming Cui, Mohamed Abdelrazek, John Grundy, Hai Jin, Qiang He, Budgeted Data Caching based on k-Median in Mobile Edge Computing, *27th IEEE International Conference on Web Services* (**ICWS2020, CORE A**), pp. 197-206, Beijing, China, 2020.

8. **Xiaoyu Xia**, Feifei Chen, Qiang He, Guangming Cui, Phu Lai, Mohamed Abdelrazek, John Grundy, Hai Jin, Graph-based Optimal Data Caching in Edge Computing, *17th International Conference on Service-Oriented Computing* (**ICSOC2019, CORE A**), pp. 477-493, Toulouse, France, 2019.

9. **Xiaoyu Xia**, Feifei Chen, Qiang He, Guangming Cui, John Grundy, Mohamed Abdelrazek, Athman Bouguettaya, Hai Jin, Formulating Cost-Effective Data Distribution Strategies Online for Edge Cache Systems, *IEEE Transactions on Parallel and Distributed Systems* (**TPDS, CORE A***), *Major Revision.*

10. **Xiaoyu Xia**, Feifei Chen, Qiang He, Guangming Cui, John Grundy, Mohamed Abdelrazek, Formulating Interference-aware Data Delivery Strategies in Edge Storage Systems, *IEEE International Conference on Computer Communications* (**INFOCOM2022, CORE A***), *Under Review.*

11. **Xiaoyu Xia**, Feifei Chen, Qiang He, Guangming Cui, John Grundy, Mohamed Abdelrazek, Cache Data Delivery on the Fly in Edge Caching Systems, *IEEE International Conference on Computer Communications* (**INFOCOM2022, CORE A***), *Under Review*.

Co-author Papers

1. Guangming Cui, Qiang He, Bo Li, **Xiaoyu Xia**, Feifei Chen, Hai Jin, Yun Yang, Efficient Verification of Edge Data Integrity in Edge Computing Environment, *IEEE Transactions on Services Computing* (**TSC, CORE A***), accepted in 2021. DOI:10.1109/TSC.2021.3090173.

2. Phu Lai, Qiang He, **Xiaoyu Xia**, Feifei Chen, Mohamed Abdelrazek, John Grundy, John Hosking, Yun Yang, Dynamic User Allocation in Stochastic Mobile Edge Computing Systems, *IEEE Transactions on Services Computing* (**TSC, CORE A***), accepted in 2021. DOI:10.1109/TSC.2021.3063148.

3. Ying Liu, Yuzhen Han, Ao Zhang, **Xiaoyu Xia**, Feifei Chen, Mingwei Zhang, Qiang He, QoE-aware Data Caching Optimization with Budget in Edge Computing, *28th IEEE International on Web Services* (**ICWS2021, CORE A**), Virtual, accepted in 2021.

4. Ying Liu, Qiang He, Dequan Zheng, **Xiaoyu Xia**, Feifei Chen, Bin Zhang, Data Caching Optimization in the Edge Computing Environment, *IEEE Transactions on Services Computing* (**TSC, CORE A***), accepted in 2020. DOI:10.1109/TSC.2020.3032724.

5. Bo Li, Qiang He, Guangming Cui, **Xiaoyu Xia**, Feifei Chen, Hai Jin, Yun Yang, READ: Robustness-oriented Edge Application Deployment in Edge Computing Environment, *IEEE Transactions on Services Computing* (**TSC, CORE A***), accepted in 2020. DOI:10.1109/TSC.2020.3015316.

6. Guangming Cui, Qiang He, **Xiaoyu Xia**, Phu Lai, Feifei Chen, Tao Gu and Yun Yang, Interference-aware SaaS User Allocation Game for Edge Computing, *IEEE Transactions on Cloud Computing* (**TCC**), accepted in 2020, DOI:10.110 9/TCC.2020 .3008448.

7. Phu Lai, Qiang He, Guangming Cui, **Xiaoyu Xia**, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy and Yun Yang, QoE-aware User Allocation in Edge Computing Systems with Dynamic QoS, *Future Generation Computer Systems* (**FGCS, CORE A**), Vol 112, pp. 684-694, 2020.

8. Zhiwei Xu, Guobing Zou, **Xiaoyu Xia**, Ya Liu, Yanglan Gan, Bofeng Zhang, Qiang He, Distance-aware Edge User Allocation with QoE Optimization, *27th IEEE International on Web Services* (**ICWS2020, CORE A**), pp. 66-74, Beijing, China, 2020.

9. Guangming Cui, Qiang He, **Xiaoyu Xia**, Feifei Chen, Hai Jin, Yun Yang, Robustness-oriented k Edge Server Placement, *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing* (**CCGrid2020, CORE A**), pp. 81-90, Melbourne, Australia, 2020.

10. Feifei Chen, Jingwen Zhou, **Xiaoyu Xia**, Hai Jin, Qiang He, *13th IEEE Conference on Cloud Computing* (**CLOUD2020, CORE B**), pp 184-192, Beijing, China, 2020.

11. Ying Liu, Ao Zhang, **Xiaoyu Xia**, Feifei Chen, Bin Zhang, Qiang He, Proactive Data Cache and Replacement in the Edge Computing Environment, *13th IEEE Conference on Cloud Computing* (**CLOUD2020, CORE B**), pp. 193-200, Beijing, China, 2020.

12. Phu Lai, Qiang He, Guangming Cui, **Xiaoyu Xia**, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, Yun Yang, Edge User Allocation with Dynamic Quality of Service, *17th International Conference on Service-Oriented Computing* (**ICSOC2019, CORE A**), pp. 86-101, Toulouse, France, 2019.

Abstract

The world is witnessing an exponential growth of mobile and Internet-of-Things devices in recent years. The enormous network traffic often causes network congestion and increases network latency. To tackle this challenge, *edge computing*, as an extension of cloud computing, has emerged as a promising paradigm for powering a variety of applications demanding low latency, e.g., virtual or augmented reality, interactive gaming, etc. In an edge computing environment, edge servers are deployed at base stations geographically to offer highly accessible computing capacities and services to nearby users. As the number of end-users accessing edge apps increases, it is expected that a large amount of app data will be transmitted via edge servers between the remote cloud server and users' end-devices. Caching app vendors' data on edge servers will significantly reduce their users' data retrieval latency because the users can retrieve data directly from edge servers within their close geographic proximity.

Data caching in edge computing differs from data caching in the cloud computing environment as well as other conventional networks with its three unique constraints, i.e., server capacity constraint, server coverage constraint and server adjacency constraint. Thus, the data caching strategies from cloud computing and conventional networks cannot be directly applied in edge computing. App vendors in edge computing face three major challenges: 1) How to cache data with limited storage resources on edge servers to provide low-latency services for their users? 2) How and when to release or replace cached data by popular ones to ensure the service quality of users and low cost in the long-term? 3) How to cost-effectively distribute required data to specific caches on edge servers? In this thesis, we formulate those challenges as three

major data caching problems in edge computing, i.e. edge data placement, edge data replacement and edge data distribution problems.

From an app vendor's perspective, caching app data on edge servers can ensure low latency in its users' data retrieval. To minimize the data caching cost and maximize the reduction in service latency, we study the individual data placement problem and propose a heuristic algorithm. Considering constrained cache spaces on edge servers due to their physical sizes, we further propose an approximation algorithm to place multiple popular data properly for the app vendor to minimize overall user latency with its reserved cache spaces on edge servers.

Unlike conventional caching systems where cache data never or seldom move, data in an edge computing environment often needs to be moved across edge servers in order to be delivered to users without violating their latency constraints. What complicates the problem further is the temporal dynamics in the real-world edge caching system. Users join and leave the system randomly, requesting different data over time. Without complete information about future dynamics, we propose a Lyapunov-based approach to place and replace data in edge servers' caches, aiming to minimize the total system cost while ensuring users' average data retrieval latency. In addition, we consider the system benefit produced by the reduction of latency and propose an online frame to formulate cost-effective edge data replacement strategies for solving this problem.

As an app vendor, app data needs to be transferred from the remote cloud server to specific edge servers in an area to serve the app users in the area. However, according to the pay-as-you-go business model, distributing a large amount of data from the cloud to edge servers can be expensive. Thus, app the vendor must minimize the cost incurred, while the data distribution must not take too long. To solve this problem, we first propose a two-phase algorithm based on Steiner Tree to solve this problem effectively and efficiently in quasi-static scenarios. Then we design an online approach based on Lyapunov Theory to solve this problem in each time slot without future information.

In summary, this thesis studies the data caching problems in edge computing, including edge data placement, edge data replacement and edge data distribution problems, from the app vendor's perspective. It makes a substantial contribution to the edge computing community and provides a new perspective to solve a huge number of problems in the edge computing environment.

Keywords: edge computing, data caching, data distribution, data placement, data replacement, optimization.

Chapter 1

Introduction

1.1 Background

The world is witnessing exponentially growing mobile traffic, which is predicted to increase at an average annual rate of 46%, and will achieve 77 exabytes per month by 2022 [19]. This produces an enormous load on networks, as considerable network resources are required to manipulate and transmit this massive data. Possible consequences include increased network latency and network congestion, which can significantly impact end-users' quality of experience. To address this issue, edge computing has emerged as a new distributed computing paradigm that allows cloud computing capacities to be distributed to *edge servers* [28]. These edge servers, each powered by one or many physical machines, are deployed at base stations that are geographically close to end-users. Vendors of Mobile and IoT application (referred to as *apps* together hereafter) can hire computing and storage resources on edge servers for hosting their apps to serve near-by app users with low latency and high throughput [48]. App users can offload computation tasks from their resource-constrained devices to nearby edge servers [9, 10, 50, 61]. It is a key technology that facilitates

the 5G mobile network [26].

As edge servers become most mobile devices' entry points to the Internet, a large proportion of the rapidly increasing mobile traffic data will be transmitted through those edge servers. Those edge servers provide the infrastructure for caching popular data requested by app users, which often accounts for a significant percentage of the mobile traffic, for example, a viral short video. If the requested data is available on a nearby edge server, an app user does not have to retrieve it from the app server in the cloud. Caching popular data on edge servers can thus considerably reduce the latency in app users' data retrieval. From an app vendor's perspective, it can also largely reduce the volume of data transferred from the cloud, which may incur substantial data transmission cost [42].

Data caching techniques have been widely employed in many domains, from hardware cache, e.g., CPU [46], GPU [38], disks [44], to software cache, e.g., web [39], database [17], etc. In the network domain, data caching has also been intensively studied for its advantages in saving bandwidth consumption, reducing network latency and so forth. In the last few years, many researchers have investigated network cache from different perspectives, e.g., cache allocation and replacement strategies [12], coded caching [27, 36], routing and caching [13], and information theoretic caching [35, 51]. As a new distributed computing paradigm, edge computing offers new opportunities and raises critical challenges for data caching. The fundamental mechanism is to cache popular data on edge servers so that nearby app users can retrieve it with low latency. This is particularly important and useful for latency-sensitive applications, e.g., video streaming, gaming, navigation, augmented reality, etc. It is predicted that mobile traffic can be reduced by 35% through caching data

on edge servers [18].

Edge computing is significantly different from cloud computing which facilitates content-centric network and content delivery network. In the edge computing environment, adjacent edge servers deployed at different base stations and access points can communicate with their neighbor edge servers and share their storage resources via high-speed links [22, 9]. App users' workloads in a particular area can be transferred and balanced across the edge servers covering that area [9]. Thus, the edge servers in a particular area constitute an *edge server network*, which can be modeled as a graph where a node represents an edge server and an edge represents the link between two edge servers. Data caching in an edge server network differs from data caching in the cloud computing environment as well as other conventional network environments with its three unique constraints, i.e.,*server capacity constraint*, *server coverage constraint* and *server adjacency constraint*.

Server Capacity Constraint: Unlike cloud servers that have access to virtually unlimited storage capacities in the cloud, storage capacities of edge servers are limited due to the physical size limit [9, 15, 40, 52, 66]. The competition between app vendors makes it impossible for an app vendor to cache all app data on every edge server. Thus, the common practice is for app vendors to reserve certain cache spaces on edge servers for caching popular app data.

Server Coverage Constraint: An edge server covers a specific geographical area so that app users within its coverage can connect to this edge server [26]. In a particular area, a number of edge servers are usually deployed in a distributed manner so that they can cover different geographical areas. Normally, the coverage areas of adjacent edge servers partially overlap to avoid blank areas not covered by any edge

servers. An app user in the overlapping area can connect to any one of the edge servers covering the app user.

Server Adjacency Constraint: An app user can retrieve a piece of app data from its nearby edge servers (referred to as *local edge servers* hereafter) that cover the app user if the app data is cached on one of these edge servers. If the app data is not cached on any of those local edge servers, the app user can retrieve it from its *neighbor edge servers* within acceptable latency [9]. Either way, it is faster than retrieving the app data from a server in the remote cloud [22].

Recently, research has started in earnest to investigate data caching problems in the edge computing environment from the infrastructure provider's perspective with different offline optimization objectives, e.g., minimum response latency [56], minimum delay cost [47], maximum data sharing efficiency [32]. However, these studies have not systematically considered the requirements and concerns of app vendors like Facebook or Uber who posses storage resources on edge servers to cache their data. To the best of our knowledge, this thesis takes the first attempt to investigate data caching problems in edge computing from an app vendor's perspective.

1.2 Research Questions

Due to those unique constraints mentioned in Section 1.1, the data caching strategies from conventional networks and cloud computing cannot be directly applied in edge computing. To solve the data caching problems in edge computing, app vendors who are important stakeholders in the edge computing environment face three major challenges represented as three research questions:

- **RQ1: How should we place data into caches with limited storage resources on edge servers to provide low-latency services for their users?** For an app vendor, caching app data on edge servers can ensure low latency in its users' data retrieval. A straightforward solution for this problem is to cache all requested data on all the edge servers in a particular area for nearby app users to access. This way, the latency in all app users' data retrieval can be minimized. However, based on the pay-as-you-go pricing model, the app vendor will need to hire substantial resources on edge servers for caching the data. This incurs excessive *caching cost* and is impractical for most, if not all, app vendors. This question can be formulated as an edge data placement problem, aiming to select suitable edge servers to cache app vendor's data in different scenarios. Thus, we should investigate the effective and efficient data place strategies with available caches to answer this question.

- **RQ2: How and when should we release or replace cached data by popular ones to ensure the service quality of users and low cost in the long-term?** In real-world scenarios, the status of a dynamic edge computing system changes over time. For example, users may leave the system and new popular data may be requested by new users. Except remaining optimal, those dynamics must be taken into account. However, those information is not available prior to their occurrences. Thus, the answer of **RQ1** cannot solve this question. In this thesis, we formulate this research question as an edge data replacement problem. To solve this problem, we should design the approaches to update cached data over time in an online manner without the need for knowing future data dynamics in the real-world edge computing system.

- **RQ3: How can we cost-effectively distribute required data to specific caches on edge servers?** Once the edge data placement or replacement strategies are decided, how to distribute the requested data is another important problem for app vendors. Unlike data transmission in cloud computing and wireless sensor networks, app data distribution in the edge computing environment consists of two major components: 1) data transmission from the cloud to edge servers; 2) and data transmission between edge servers. Both components must be considered in a systematic manner to formulate cost-effective data distribution strategies for app vendors. Similar to **RQ2**, the dynamics should also be considered in formulating data distribution strategies. To answer this question, we should propose the cost-effective data distribution strategies with consideration of the unique constraints and dynamics of edge computing.

1.3 Significance

First proposed by Cisco in 2012 [5], edge computing facilitates a new distributed computing paradigm. In terms of the network topology and infrastructure deployment, edge computing extends cloud computing by distributing computing resources and services from the central cloud servers or hosts to the geographically distributed edge servers in the network. The users of a variety of apps can benefit from the advantages provided by edge computing, including video streaming, real-time navigation, gaming as well as many IoT applications [60]. Edge computing is also a key technology of the 5G mobile network [59].

Being a new technology, researchers and practitioners are working on standardization to ensure the technology is robust, open, and secure, which is critical to large

scale deployments of edge systems. One of the first cities, Barcelona in Spain, has implemented edge computing with more than 3,000 edge servers deployed across the city serving thousands of IoT devices [60]. With edge servers deployed within an area, many issues of computation offloading and data caching have been raised in the edge computing environment.

Edge computing inherits the pay-as-you-go pricing model from cloud computing, which allows app vendors to hire storage resources on edge servers from edge infrastructure providers to cache app data for their own users. Thus, both the benefit produced and the cost incurred by data caching for app vendors is critical to the success of edge computing because, after all, app vendors are the main customers in the edge computing environment. Unfortunately, all the existing work on data caching in the edge computing environment tackles the data caching problem from either the users' or the edge infrastructure providers' perspectives, none from the app vendors' perspective as to how to cost-effectively cache data in the edge computing environment.

Our research aims at providing data placement, data replacement and data distribution strategies in edge computing for app vendors that can save them a significant amount of costs while guaranteeing high quality of services with low latency. The major contributions of this thesis are as follows:

1. **The first attempt to investigate the edge data placement problem from an app vendor's perspective** and to propose effective and efficient approaches to solve those problems.

2. **The first attempt to solve the edge data replacement problem** by

designing online approaches for app vendors in dynamic edge computing environments.

3. **The first attempt to help app vendors save their data transmission costs** by solving the edge data distribution problem in both quasi-static and dynamic edge computing scenarios.

1.4 Research Methodology

Our research is systematically conducted from the ground up. Several key steps have been and will be adopted in our research:

- **Literature review**: The purpose of a literature review is to discover what is known and what is unknown. We have been conducting a comprehensive literature review in the areas of data caching in conventional networks, cloud computing and edge computing. Firstly, we try to identify (1) what research problems in cloud computing also exist in edge computing but have not been addressed, or poorly addressed; (2) new research problems in edge computing; and (3) existing approaches of solving relevant problems. To the best of our knowledge, our work is the first to tackle the data caching problem in edge computing scenarios from app vendor's perspective with multiple edge servers, end-users and different constraints.

- **Solution design**: As most tasks in our research problems are optimization problem, we use a variety of optimization techniques to solve ours research problems, including Integer Programming, Lyapunov Optimization, Game Theory

and Prime Dual approach. Furthermore, after a thorough review of the literature in relevant areas, we gained big picture of the latest state-of-the-art approaches that are being adopted. Then we analyzed the pros and cons of those approaches, selected the most suitable approaches, and improved on them if necessary.

- **Experimental and Simulation methods**: We conducted a series of experiments with real-world dataset to simulate the network and verify the effectiveness and efficiency of our approaches. Our experiments are extensive and cover many different scenarios that might exist in a real-world setup, namely different numbers of edge servers, different numbers of users, varying edge densities, differentiated budgets and storage reservations.

1.5 Thesis Outline

This section presents the structural organization of this thesis. According to three research problems addressed in this thesis, i.e., edge data placement, edge data replacement and edge data distribution, the chapters are organized as follows.

1.5.1 *Chapter 2 Literature Review*

This chapter provides an overview of prior studies on data caching problems in the fields of conventional networks, cloud computing and edge computing. Due to the unique constraints of edge computing, including server capacity, coverage and adjacency constraints, data caching strategies from conventional networks and cloud

computing are not suitable to solve data caching problems in edge computing. Moreover, existing studies on data caching problems in edge computing are from the user's or edge infrastructure provider's perspective. According to the literature review, this thesis makes first attempt to tackle the data caching problems in edge computing for app vendors.

1.5.2 *Chapter 3 Edge Data Placement*

This chapter makes the first attempt to investigate the edge data placement problems from an app vendor's perspective. This chapter answers **RQ1**. **Chapter 3.1** investigates the individual data placement problem in edge computing and has been published on *Future Generation Computer Systems*. In **Chapter 3.1**, we first formulate this problem as a constrained optimization problem with the aim to serving all users with the minimum caching cost. We solve this problem by Integer Programming and a greedy algorithm. Considering that app vendors usually have multiple data to be place and cache spaces on edge servers are usually limited, we further formulate the edge multi-data placement problem in **Chapter 3.2**, aiming to make the maximum latency reduction by placing data onto reserved cache spaces. To solve this NP-hard problem, we design an approximation algorithm based on the weighted k-set packing problem. **Chapter 3.2** has been published on *IEEE Transactions on Services Computing*.

1.5.3 *Chapter 4 Edge Data Replacement*

Over time, new users may arrive and existing users may depart in the edge computing environment. Data popularity may also dynamically change. However, the

approaches proposed in **Chapter 3** cannot handle such events. To fill this gap and answer **RQ2**, **Chapter 4** studies the edge data replacement problem, considering the dynamics in edge computing. In **Chapter 4.1**, we formulate this edge data replacement as a dynamic system in long-term, aiming to minimize the system cost of app vendors, while ensuring the low latency of users. To achieve this, we propose an online algorithm based on Lyapunov optimization. This work has been published on *IEEE Transactions on Parallel and Distributed Systems*. However, except the cost, the benefit produced by placing and replacing data is also important to the success of app vendors. Thus, **Chapter 4.2** further studies this edge data replacement problem with consideration of cost-effectiveness. We propose an online frame based on two heuristic algorithms to solve this cost-effective edge data replacement problem, with higher effectiveness and efficiency compared with the approach proposed in **Chapter 4.1**. This work has been published on *IEEE Transactions on Mobile Computing*.

1.5.4 *Chapter 5 Edge Data Distribution*

Once app vendors make decisions for placing or replacing data on the distributed edge servers, each data should be transmitted to the destination edge servers, i.e. the edge servers to cache this data, from either the remote cloud server or a source edge server that has already cached this data. According to the pay-as-you-go pricing model, app vendors must pay for the transmission cost incurred in this process. **Chapter 5** studies the edge data distribution problem in both quasi-static and dynamic edge computing scenarios to answer **RQ3**. In **Chapter 5.1**, we study this edge data distribution problem from the app vendor's perspective, with the aim to minimize the data transmission cost while ensuring the low latency incurred during the data

distribution process. To solve this problem, we design a two-step approximation algorithm based on the Steiner tree problem. The work presented in **Chapter 5.1** has been published on *IEEE Transactions on Parallel and Distributed Systems*. Similar to the issue in **Chapter 3**, the dynamics is not considered in **Chapter 5.1**. Thus, we study the online edge data distribution problem in **Chapter 5.2** by using Lyapunov optimization to deal with the dynamics in the EC environment. This work currently is under major revision for *IEEE Transactions on Parallel and Distributed Systems*.

1.5.5 *Chapter 6 Conclusions and Future work*

In this chapter, we first summarize the contributions of this thesis, including the problems studied and the approaches proposed in **Chapters 3 - 5**. After that, we discuss the potential research problems based on this thesis, as the future work.

Chapter 2

Literature Review

Data caching have been extensively investigated in the fields of conventional networks and cloud computing. With the popularity of edge computing, data caching in the edge computing environment is obtaining a lot of attentions from researchers in the last few years.

2.1 Data Caching in Conventional Networks and Cloud Computing

Many Data caching issues in the conventional network environment has been extensively investigated in the last few decades, e.g., web caching [39], content-centric networking [55], publish-subscribe systems [45], content delivery network [4], etc. To name a few representative pieces of work, Banerjee et al. [3] developed a content placement strategy for information-centric network based on data popularity, namely Greedy Caching. With popular contents cached in the network, the Greedy Caching approach determined whether the contents should be cached in the core server based on the cache miss rate at the edge of the network. In [49], the authors formulated

two caching strategies for data publish-subscribe systems, including eviction-based caching and time-to-live-based caching to address the space and time issues respectively. The authors of [30] focused on balancing the trade-off between latency and cost in the content-centric network. They addressed this issue with a holistic model for provisioning the storage capability based on the network performance and the provisioning cost. Shafiq et. al[43] studied the characteristics of caching workload in content delivery network. They designed the measurement and analysis methods for evaluating data caching replacement algorithms based on the network workload logs.

In the cloud computing environment, a critical data caching problem is how to utilize cache space efficiently on cloud hosts and mobile devices. Arteaga et al. [1] proposed an on-demand cache management method, namely CloudCache, to fulfill the caching requirement of the workload and minimize cache wear-out. They also presented a dynamic cache migration solution to cache capacity balance across cloud hosts by live cached data migration. In [33], the authors presented how to use segment access-aware dynamic semantic cache for relational databases in the cloud environment. A cache access algorithm was introduced that considers cache exact hit, cache extended hit, cache partial hit and cache miss. In [21], the authors leveraged the ability of collaborative edge servers for minimizing app vendors' data caching cost. They proposed an online algorithm to determine how data should be retrieved, placed and replaced to fulfill users' data requests. The authors of [2] explored the cache design space for embedded processors with evolutionary techniques for mobile and thin client processors in the cloud computing environment. A heuristic and evolutionary method was presented to generate a near-optimal cache space design for enhancing service quality. In [24], the authors formulated a benefit maximization problem and

created a cache replacement approach based on spatio-temporal traffic requirements. They also introduced a content clustering method for collecting popular data and clustering similar contents. Halalai et al. [23] proposed Agar, a caching system, by implementing erasure-code into data caching techniques. Considering the data popularity and network latency, Agar could find the optimal solution to cache data chunk by dynamic programming.

However, the data caching strategies from conventional networks and cloud computing cannot be directly applied in edge computing, due to the unique constraints including server capacity constraint, server coverage constraint and server adjacency constraint.

2.2 Data Caching in Edge Computing

From the perspectives of network topology and infrastructure deployment, edge computing is an extension of cloud computing with distributed computing capacities and services at the edge of the network. App users in various domains can benefit from the advantages of edge computing, e.g., interactive gaming, real-time navigation, augmented reality [60]. Offering many unique advantages, edge computing also raises various new research challenges from the service provider's perspective, e.g., edge user allocation [25, 28], edge computation offloading [11], etc.

Recently, some researchers have begun to tackle the challenges of data caching in the edge computing environment. As mentioned in Section 1, there are three unique constraints in the edge computing environment, which are not considered by both cloud computing and conventional networks. In this case, conventional approaches

Table 2.1: Summary of Data Caching Studies in Edge Computing

Proposals		[58]	[6]	[29]	[65]	[40]	[53]	[34]	[64]	[8]	[57]	[16]	[62]	[67]	[14]	[47]	[37]	This thesis
Independent Research									✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Scenarios	Quasi-static		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓					✓
	Dynamic	✓					✓							✓	✓	✓	✓	✓
Objectives	Latency	✓	✓	✓	✓			✓	✓		✓	✓	✓	✓				✓
	Cost			✓						✓					✓	✓	✓	✓
	Workload		✓			✓	✓	✓			✓							
	Energy Consumption	✓																
Techniques	Integer Programming							✓								✓		✓
	Primal-Dual														✓			
	Heuristic		✓	✓	✓	✓		✓	✓	✓	✓	✓				✓	✓	✓
	Game Theory				✓								✓					✓
	Lyapunov Theory	✓					✓							✓				✓

for data caching are not suitable in edge computing and cannot be applied directly. Thus, new ideas and techniques are being proposed and investigated. The reviewed literature are summarized in the table above, including the scenarios, objectives and techniques in existing related studies as well as this thesis.

2.2.1 Joint Optimization Studies with Data Caching in Edge Computing

In [58], the authors studied the joint data caching and task offloading problem in the dense mobile edge computing environment. Considering the dynamics of edge computing, the authors formulated this problem as a dynamic system, with the aim to to minimize the total network latency and applies a long-term energy consumption constraint to stabilize the edge caching system. To solve this problem, they treated the energy of each device as a queue and proposed an online algorithm, named OREO, based on Lyapunov optimization technique.

Breitbach et al. [6] considered the joint optimization problem of data caching and task scheduling for IoT applications in the edge computing environment. In this paper, the authors proposed a data management system for edge computing environments by decoupling data placement for task scheduling, with the consideration of multiple context dimensions. The proposed system could adjust the data replica placement cost to achieve the balance between data management overhead and execution delay.

Li et al. [29] also studied a joint optimization problem of data caching and task scheduling in edge computing. The authors aimed to minimize the response time and computational overheads, while improving users' quality of experience (QoE).

To formulate this problem, the authors investigated the task priority, the relevance between edge servers and tasks and the cost incurred during the transmission process. They proposed a greedy algorithm, embedded an existing Kuhn–Munkres algorithm, to solve this problem efficiently.

The authors of [65] integrated in-network caching and edge caching to guarantee the quality of time-sensitive multimedia transmissions over the 5G wireless network. They provided three hierarchical edge caching mechanisms, including a random hierarchical caching approach, a proactive hierarchical caching approach and a game-theory-based hierarchical caching approach.

Poularakis et al. [40] studied the joint optimization of data placement and request routing in the edge computing environment, aiming to minimize the load of the cloud. The authors first formulated this joint problem as a constrained optimization problem and studied several cases of this problem. After that, they proposed a randomized rounding algorithm to find a close-to-optimal solution to this problem and proved the approximation ratio of the proposed algorithm.

In [53], the authors integrated the cloud radio access network with the edge computing technology to schedule resources including caches and computational resources dynamically. To achieve a balance between computation resource usage and network resource usage, they formulated this problem as a stochastic problem. With consideration of variable task request lengths, the authors proposed the VariedLen algorithm, based on Lyapunov optimization, to maximize the mobile network provider's profit in the edge computing environment.

The authors of [34] investigated the joint data caching and workload scheduling problem in the mobile edge computing environment. They aimed to minimize the

transmission delay and the data traffic to cloud based on the collaborations among edge servers when making edge data placement decisions. They first formulated this problem as a mixed integer nonlinear problem. Then they proposed a two-phases heuristic approach, named ICE, based on Gibbs sampling to solve this problem within a reasonable time.

However, the above studies investigate edge data caching only to complement computing offloading and task offloading, and fail to give data caching sufficient attention as a unique technology with advantages in reducing data retrieval latency and improving the quality of services and users' experiences.

2.2.2 Offline Data Caching Studies in Edge Computing

In the recent years, some researches have started to investigate the edge data placement problem itself, rather than a complement of computing offloading and task offloading. The authors of [64] proposed a hierarchical caching mechanism in the edge computing environment with consideration of wireless communication. They aimed to maximize the hitting rate by placing data in different layers including routers, base stations and mobile devices, while ensuring the data transmission latency in a low level. To solve this problem, the authors introduced a simple but effective heuristic approach with theoretical guarantee.

In [8], the authors studied a content-aware data placement problem in mobile edge computing, from the mobile network operator's perspective. They introduced an optimal auction mechanism, considering the based on cache allocation and content priority from user valuation reports. To find the optimal solution to this problem, the

authors proposed computationally-efficient approaches to apply the auction mechanism based on data retrieval and delivery costs for calculating the optimal decisions of data placement.

Xie et al. [57] investigated the data placement problem in the edge computing environment, with consideration of data retrieval process. They aimed to reduce both data retrieval latency and implementation overhead while balancing the loads of edge servers. To achieve those aims, GRED, an efficient edge data placement algorithm, is proposed to balance the data retrieval workloads across the edge computing system and shorten the path for delivering data to users.

In [16], Drolia et al. provided Cachier, a caching system to minimize the latency of data retrieval by data placement for latency-sensitive image recognition applications in the edge computing environment. Cachier considered the locality, cache size, cache replacement policy and request estimation. In addition, a coordinating mechanism was applied in Cachier for balancing the work loads between edge servers and the remote cloud server.

Zhang et al. [62] proposed a cooperative edge caching architecture for enhancing edge data placement with the computation resources on edge servers. In this mobility-aware architecture, the authors implemented vehicles to delivery caching contents into the caches on edge servers. In addition, those vehicles were used as edge servers to provide external caches. The authors used a game theoretic approach to achieve a Nash equilibrium for finding a feasible solution.

However, the above studies only investigate the edge data caching problem in an offline manner without considering the user mobility and temporal data dynamics in real-world edge cache systems - users' demands on data vary at different locations

over time.

2.2.3 Online Data Caching Studies in Edge Computing

Instead of solving the edge data placement problem optimally in an offline manner, some researchers are investigating online approaches for solving the dynamic edge data placement problems, i.e. edge data replacement problem.

In [67], the authors studied the budgeted data replacement problem in the edge computing environment. Considering the limited resources on edge servers, the authors aimed to minimize the long-term overall latency of all users under a budget consisting of the cache cost, computation cost and traffic cost. The authors proposed an online algorithm based on Lyapunov optimization to solve this problem in each time slot.

Deng et al. [14] also investigated the data replacement problem in the edge computing environment but tried to minimize the overall cost rather than latency. They involved the request life cycle and billing model in this problem. They provided an online primal-dual algorithm named IDA4ReE to solve this problem under a resource constraint and performance requirement.

Tran et al. [47] targeted edge video update strategies specifically in the edge computing environment. Based on video request scheduling, they aimed to achieve the minimum access delay cost with consideration of capacity constraints of edge servers. They first formulated this problem as an Integer Programming problem, and then proposed an heuristics-based approach for video data placement to optimize users' quality of experience by placing and replacing different bitrate versions of a video on edge servers.

In [37], the authors studied the edge data replacement problem from the user's perspective. They aimed to provide a personalized data placement management based on user preference to minimize the overall cost including the data migration cost and perceived latency cost. To solve this problem, the authors first formulated this problem as a contextual Multi-armed Bandit problem, and then proposed an online approach based on Thompson-sampling to detect the dynamics in the edge computing environment.

Edge computing inherits the pay-as-you-go pricing model from cloud computing, which allows app vendors to hire storage resources on edge servers from edge infrastructure providers to cache app data for their own users. Thus, both the benefit produced and the cost incurred by data caching for app vendors is critical to the success of edge computing because, after all, app vendors are the main customers in the edge computing environment. Unfortunately, all the above existing work on data caching in the edge computing environment tackles the data caching problem from either the user's or the edge infrastructure provider's perspective, none from the app vendors' perspective as to how to cost-effectively cache app data the edge computing environment. To the best of our knowledge, we make the first attempt in this thesis to tackle the data caching problems in edge computing scenarios for app vendors.

2.2.4 Summary

This chapter surveys data caching problems in various scenarios, including conventional networks, cloud computing and edge computing. Data caching strategies from conventional networks and cloud computing are not applicable to solve data caching problems in edge computing, due to the unique constraints of edge computing, i.e.

server capacity constraint, server coverage constraint and server adjacency constraint. Edge computing inherits the pay-as-you-go pricing model from cloud computing. Thus, app vendors are critical to the success of edge computing because they are the main users of the edge servers. However, existing data caching strategies in edge computing are investigated from either the edge infrastructure provider or the user's perspective. To fill this knowledge gap, this thesis defines three major research problems, including edge data placement, edge data replacement and edge data distribution problems, and tackles those problems in edge computing scenarios from the app vendor's perspective.

Chapter 3

Edge Data Placement

Edge computing has emerged as a new computing paradigm that allows computation and storage resources in the cloud to be distributed to edge servers. Those edge servers are deployed at base stations to provide nearby users with high-quality services. Thus, data placement is extremely important in ensuring low latency for service delivery in the edge computing environment. This chapter is comprised of two published papers, aiming to solve the individual data and multi-data placement problems for app vendors in the edge computing environment.

3.1 Individual Data Placement Strategies in Edge Computing

Given a piece of the app vendor's popular data, a straightforward strategy is to place this data on each edge server. This way, the latency in all app users' data retrieval can be minimized. However, edge computing, as an extension of cloud computing, also employs the pay-as-you-go pricing model. App vendors need to hire storage resources on edge servers for placing this data. This solution produces a huge *caching cost*, and thus is impractical for most, if not all, app vendors. To minimize the cost and maximize the latency reduction, we formulate this individual data placement problem as a constrained optimization problem in this chapter, and prove that this problem is NP-complete. We first propose an optimal solution named IPEDC to solve this problem based on Integer Programming, and then provide a greedy algorithm named AEDC to find near-optimal solutions. We conduct intensive experiments on a real-world data set and a synthesized data set to evaluate our approaches. Our results demonstrate that IPEDC and AEDC significantly outperform the four representative baseline approaches.

This chapter is based on a published paper, entitled: Graph-based Data Caching Optimization in Edge Computing, *Future Generation Computer Systems*, Vol. 112, pp. 684-694, 2020.

Future Generation Computer Systems 113 (2020) 228–239

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Graph-based data caching optimization for edge computing

Xiaoyu Xia [a], Feifei Chen [a], Qiang He [b,*], Guangming Cui [b], Phu Lai [b], Mohamed Abdelrazek [a], John Grundy [c], Hai Jin [d]

[a] Deakin University, Geelong, Australia
[b] Swinburne University of Technology, Hawthorn, Australia
[c] Monash University, Clayton, Australia
[d] Huazhong University of Science and Technology, Wuhan, China

ARTICLE INFO

Article history:
Received 12 December 2019
Received in revised form 29 May 2020
Accepted 6 July 2020
Available online 13 July 2020

Keywords:
Optimization
Edge computing
Edge data caching

ABSTRACT

Edge computing has emerged as a new computing paradigm that allows computation and storage resources in the cloud to be distributed to edge servers. Those edge servers are deployed at base stations to provide nearby users with high-quality services. Thus, data caching is extremely important in ensuring low latency for service delivery in the edge computing environment. To minimize the data caching cost and maximize the reduction in service latency, we formulate this *Edge Data Caching (EDC)* problem as a constrained optimization problem in this paper. We prove the $\mathcal{NP}$-completeness of this EDC problem and provide an optimal solution named IPEDC to solve this problem based on Integer Programming. Then, we propose an approximation algorithm named AEDC to find approximate solutions with a limited bound. We conduct intensive experiments on a real-world data set and a synthesized data set to evaluate our approaches. Our results demonstrate that IPEDC and AEDC significantly outperform the four representative baseline approaches.

1. Introduction

The world has witnessed an exponential growth of mobile devices including mobile phones, wearable devices, tablets, smart vehicle, and Internet-of-Things (IoT) devices [1]. These devices introduce massive traffic that leads to network congestion and significantly impacts the quality of service, especially service latency. To address this issue, cloud data caching was introduced to allow users to access high-demand data, and utilize cloud computing's configurable and powerful capacities [2].

However, with the growing demand for high-quality data and lower latency, the cloud model falls short of those requirements, due to the usually unpredictable network latency and expensive bandwidth [3]. As an evident weakness of the cloud computing paradigm, it is extremely hard to reduce delay at the wide-area network scale. *Edge computing* is proposed as a new computing paradigm to tackle this challenge, where *edge servers* are attached to base stations or access points close to users to offer them computation and storage resources at the edge of the network [4]. This way, mobile and IoT app vendors (together referred to as *app vendors* hereafter) can rent computation and storage resources in the edge computing environment to host their services and cache their data on edge servers. This way, their app users can access those services or data with low latency [5]. Edge computing is also a key technology in the 5G mobile network [6].

As edge servers become the entry (first access) point for most mobile and IoT devices, the rapidly increasing internet traffic data will be transmitted through those edge servers. Caching data, especially popular data, on edge servers will significantly reduce the transmission latency in users' data retrieval. This is particularly critical for latency-sensitive applications, e.g., smart city deployment, real-time traffic navigation systems, augmented reality applications, etc. As popular data account for a large portion of internet traffic, caching popular data on edge servers can also reduce the pressure on the backbone network. It is predicted that mobile traffic will be reduced by 35% through caching data on edge servers. From an app vendor's perspective, caching data on edge servers can decrease the volume of data transferred in and out of the cloud to save the transfer costs considerably.

Given a piece of popular data, the straightforward solution is to cache it on every edge server in a specific geographic area. This way, the latency in all app users' data retrieval can be minimized. However, edge computing, as an extension of cloud computing, also employs the pay-as-you-go pricing model. App vendors need to hire storage resources on edge servers for caching this data. This solution produces a huge *caching cost*, and thus is impractical

* Corresponding author.
E-mail addresses: xiaoyu.xia@deakin.edu.au (X. Xia), feifei.chen@deakin.edu.au (F. Chen), qhe@swin.edu.au (Q. He), gcui@swin.edu.au (G. Cui), tlai@swin.edu.au (P. Lai), mohamed.abdelrazek@deakin.edu.au (M. Abdelrazek), john.grundy@monash.edu (J. Grundy), hjin@hust.edu.cn (H. Jin).

https://doi.org/10.1016/j.future.2020.07.016
0167-739X/

for most, if not all, app vendors. Thus, app vendors must find a data caching strategy to guarantee that all their app users can access the data from nearby edge servers with low latency while minimizing the cost of hired cache spaces on those edge servers. In this paper, this data caching problem in edge computing is referred to as the *edge data caching* (EDC) problem. Our previous work [7] is the first attempt to investigate this EDC problem from app vendors' perspective. This paper significantly extends [7] by providing a more thorough theoretical analysis and a new approximation algorithm to solve the EDC problem efficiently within trusted performance bounds.

In this paper, our major contributions are as follows:

- We formulate the EDC problem from the app vendors' perspective, then prove its $\mathcal{NP}$-completeness.
- We develop an optimal approach named IPEDC for finding optimal solutions to EDC problems with the Integer Programming technique.
- We develop an approximation approach named AEDC for finding near-optimal solutions to EDC problems in large-scale scenarios efficiently, and analyze its theoretical approximation ratio.
- We conduct extensive experiments on both a real-world data set and a synthesized data set to evaluate the proposed approaches against four representative approaches.

The rest of the paper is organized as follows. Section 2 presents an example to illustrate and motivate the EDC problem. Section 3 formulates the EDC problem and proves its $\mathcal{NP}$-completeness. Section 4 presents and analyzes our optimal approach and approximation approach for finding solutions to EDC problems. Section 5 experimentally evaluates the proposed approaches. Section 6 reviews the related work. Section 7 concludes this paper and points out our key future work.

2. Motivating example

Video data is a typical type of data to be cached on edge servers. According to Cisco's report, mobile video data currently accounts for more than half of the world's mobile data traffic and it will further increase by 78% by 2021 [8]. Currently, most app vendors such as YouTube store their videos on cloud servers. When a video becomes viral over the Internet – which can result in hundreds of thousands if not millions of requests – very large numbers of users will send their requests to the cloud server for this video simultaneously. This creates tremendous traffic load on the network and increases data retrieval latency. Caching these videos on selected edge servers can effectively solve these problems.

However, in edge computing, three unique constraints differentiate the EDC problem from the data caching problems in the cloud computing environment and conventional networks:

- *Server adjacency constraint*: In the edge computing environment, edge servers can communicate their neighbor edge servers via high-speed links [9]. Thus, connected edge servers can share their computation and storage resources. This way, the edge server network can be treated as a graph where edge servers are represented by nodes and the links between edge servers are represented by edges.
- *Server coverage constraint*: To avoid any blank coverage areas in a specific geographic area, the coverage areas of nearby edge servers often intersect. Thus, app users in an overlapping area can access any of the edge servers covering them.
- *Server capacity constraint*: Different from the virtually unlimited computation and storage resources available in the cloud, edge servers only have limited computation and storage resources due to their limited sizes [10,11].

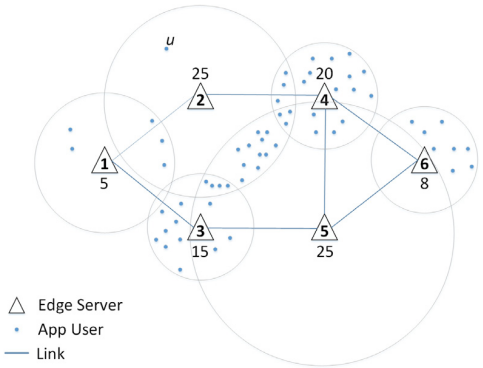


Fig. 1. An example EDC scenario.

Caching the data blocks of a data/file on multiple servers or machines is common in a large-scale cloud data center. This improves data reliability. However, in the edge computing environment, edge servers are attached to base stations that are geographically distributed. Each edge server covers the set of users within the coverage areas. Although the users within multiple edge servers' overlapping coverage area may be able to access multiple edge servers, it is usually not the case for most users. In addition, retrieving multiple data blocks from multiple edge servers and composing these data blocks into a data/file is too time-consuming in the edge computing environment where low latency is a top priority. In this paper, we have made the assumption that edge data are always cached in whole for the above reasons.

An EDC example is shown in Fig. 1. There are six edge servers in this area, with each edge server having a specific coverage area. The number next to an edge server indicates the number of users covered by that edge server. When a YouTube video becomes viral, it is predicted that a large number of YouTube users in this area will request this video. As a large amount of research effort has been made to predict video popularity [12], we assume that the number of YouTube users who will request this popular video can be predicted in this work. From YouTube's perspective, a straightforward solution to the EDC problem in this area is to cache this video on every edge server. This way, all YouTube users can access this video from edge servers. However, YouTube will need to pay for the hired resources on edge servers, such as bandwidth and storage, to cache this video. As even short videos are large, this solution is not cost-effective. Therefore, the data caching strategy must achieve the minimum data caching cost while ensuring that all the app users in this area can retrieve the video from one of the edge servers. This edge data caching (EDC) problem is inherently a Constrained Optimization Problem (COP).

The data retrieval latency and data caching cost can be evaluated using a variety of metrics. A user's data retrieval latency consists of two components: the latency between the user and its nearby edge server, and the latency between edge servers. The first component is not affected by the data caching strategy, and it is also quite small. Thus, this component is not included in the formulation of the EDC strategy. To model the COP in a more generic manner, including constraints and the optimization objective, we use the number of cached data replicas to measure the data caching cost, and the number of hops to measure the data retrieval latency. For example, the data caching cost is 6 if the video is cached on all the edge servers in Fig. 1. The *server adjacency constraint* requires that all the users must be able to retrieve the data from an edge server within one hop. For

Table 1
Summary of notations.

Notation	Description
b_u	Maximum benefit for user u
$b_{u,j}$	Benefit of caching replica on server v_j for app user u
CU	Set of users covered by the selected edge server set S
cu_i	Set of users covered by edge server v_i
$d_{i,j}$	Distance from server v_i to server v_j
d_T	Threshold of distance
d_u	Minimum distance from app user u to retrieve replica
$E = \{e_1, e_2, \ldots, e_m\}$	Finite set of links between edge servers
G	Graph presenting a particular area
$R = \{r_1, r_2, \ldots, r_n\}$	Set of binary variables indicating cache replicas on edge servers
S	Set of selected servers to cache data replica
$U = \{u_1, u_2, \ldots, u_k\}$	Finite set of users
$V = \{v_1, v_2, \ldots, v_n\}$	Finite set of edge servers

example, this constraint holds for the u in the top left corner if the video is cached on v_1, v_2 or v_4 and it does not hold if the video is only cached on v_3, v_5 and/or v_6. The rationale behind this constraint is that edge servers can communicate with their neighbor edge servers [9], but they are not designed or linked to route (potentially large) data across multiple hops. Based on the generic metrics for data caching cost and data retrieval latency, specific pricing policies and latency models can be integrated into our COP model. For example, app vendors can easily implement their own cost models and data sizes into our model.

There might exist multiple data caching solutions satisfying the latency constraint with the minimum cost. As edge servers often have different coverage radius and different user densities within their coverage areas, they usually cover different numbers of app users. Thus, the data caching solution should also reduce the maximal latency across all app users in this area. From YouTube's perspective, another optimization objective is thus to maximize the benefit produced by the cached data replicas, which is measured by the total reduction in the data retrieval latency for all the app users.

In this paper, we study quasi-static scenarios where users will not move across edge servers' coverage areas during the period of time when the EDC problem is being solved [6,9,13,14]. In highly mobile scenarios where users move across edge servers' coverage areas quickly, the app vendor can update its EDC strategy periodically or on-demand with a highly efficient EDC approach, e.g., AEDC as proposed in Section 4.2 and analyzed in Section 5.5. The model and approaches proposed in this paper are generic and applicable to various edge computing scenarios. Thus, data are cached on edge servers as a whole and we currently do not consider the situation where data can be partially cached, e.g., video segments. Also, the scale of the EDC problem in real-world scenarios can be much larger than the example presented in Fig. 1. Finding an optimal solution to a large-scale EDC problem is far from trivial.

3. Problem formulation

3.1. Problem statement

The edge servers in a particular area constitute an *edge server network*, which can be modeled as a graph where a node represents an edge server and an edge represents the link between two edge servers. Denote $G(V, E)$ as the graph, where V is the set of nodes in G and E is the set of edges in G. *In the remainder of this paper, we will speak inter-changeably of an edge server and its corresponding node in graph G, denoted as* v. The notations adopted in the paper are summarized in Table 1.

As mentioned in Section 2, this EDC problem is formulated in a generic manner: (1) using the number of data replicas to measure the data caching cost; and (2) using the number of hops between edge servers to measure the latency.

Based on the *server capacity constraint* in Section 2, edge servers only have limited resources. However, those limited resources are needed by many app vendors at the same time to host their services and cache their data for their app users. Thus, it is unlikely for one app vendor to hire most of those resources on an edge server and cache a huge amount of its data. A more cost-effective and realistic method is to cache the most popular data only for most app vendors. Therefore, this work considers the individual data caching scenarios, and builds the foundation for more sophisticated edge caching scenarios, e.g., caching multiple data.

Given a piece of data and a set of edge servers v_i ($i = \{1, \ldots, n\}$). Let $r_i \in \{0, 1\}$ be the decision indicating whether the data is cached on v_i, such that $r_i = 1$ if edge server v_i is selected to cache data. Denote the vector $R = \langle r_1, \ldots, r_n \rangle$ as the data caching solution.

The distance between two nodes in the graph can be calculated by their shortest path. As we use the number of hops to measure the data retrieval latency, the latency of an app user u can be calculated as follow:

$$d_u = \min\{d_{i,j}, r_j = 1, v_j \in V\}, \forall u \in U_i \tag{1}$$

where U_i is the set of users covered by edge server v_i.

The main objective of edge data caching is to provide high-quality services for app users with low latency. Thus, the data caching strategy R must satisfy the *server latency constraint* — the required data must be accessible from an edge server in this edge server network for every app user within a certain number of hops:

$$d_u < d_T, \forall u \in U_i \tag{2}$$

where d_T is the threshold of latency, measured by the number of hops as well.

Based on the *server adjacency constraint* discussed in Section 2, communications only occur between connected edge servers. Thus, d_T should be 2 here. However, this threshold can be relaxed if new techniques occur to allow data transmissions through multiple edge servers rapidly and the app vendor can accept the relatively high latency.

3.2. Data Caching benefit

To evaluate and compare the effectiveness of different data caching strategies, the concept of data caching benefit is introduced here, which can be calculated based on the latency reduction of user data retrieval. We use the number of hops reduced by cached data on an edge server to measure the data caching benefit. Thus, there is a negative correlation between data retrieval latency and data caching benefit. The following equation

shows how to calculate the benefit $b_{u,j}$ produced for app user $u \in U_i$ if edge server v_j is selected to cache data:

$$b_{u,j} = \begin{cases} d_T - d_{i,j} & \text{if } d_{i,j} < d_T \\ 0 & \text{if } d_{i,j} \geq d_T \end{cases} \tag{3}$$

As discussed in Section 2, to avoid the blank area that is not covered by any edge servers, the coverage of nearby edge servers often partially overlap. An app user in the overlapping area can access multiple edge servers, and retrieve data from any of those edge servers that the data in their cache. Thus, the data caching benefit produced by the data caching strategy for an app user u is:

$$b_u = \max\{r_j * b_{u,j}, v_j \in V\} \tag{4}$$

The data caching cost is the primary optimization objective from the app vendor's perspective. Thus, the data caching solution R must minimize this cost:

$$\text{minimize } cost(R) \tag{5}$$

With the minimum data caching cost, the optimal data caching strategy R should also maximize the data caching benefit:

$$\text{maximize } benefit(R) \tag{6}$$

In this way, we formulate this EDC problem as a constrained optimization problem.

3.3. Problem hardness

Here we prove the $\mathcal{NP}$-completeness of the EDC problem by Theorem 1.

Theorem 1. *The EDC problem is $\mathcal{NP}$-complete.*

Proof. To prove the $\mathcal{NP}$-completeness of the EDC problem, the minimum dominating set problem (MDS), one of the classic $\mathcal{NP}$-complete problems, is introduced first. Denote $G = (V, E)$ as an undirected graph where V is the set of n nodes and E is the set of m edges. Let $Con_{n,n}$ be the matrix to describe the connection between nodes such that $Con_{i,j}$ is 1 if there is an edge between node v_i and v_j. Denote S as the solution of this MDS problem. The MDS problem can be formulated as below:

$$\min \sum_{i=1}^{n} v_i \tag{7a}$$

$$s.t. : v_i \in \{0, 1\}, \forall i = \{1, .., n\} \tag{7b}$$

$$\sum_{i=1}^{n} Con_{i,j} \geq 1, \forall j \in \{1, \ldots, n\} \tag{7c}$$

Now we present the reduction process from the EDC problem to the MDS problem. The reduction consists of two parts: (1) making each app user only covered by one edge server; and (2) making only one app user in each edge server's coverage. Based on the reduction, the value of the benefit objective (6) is always same if selecting the same number of edge servers. In this case, the benefit objective is safely ignored. The instance $EDC(R, E, Ben_{n,k})$ can be constructed with the above reduction by an given instance $MDS(S, E, Con_{n,n})$ in polynomial time, where $n = k$ and $|R| = |S|$. In this $EDC(R, E, Ben_{n,k})$, the benefit matrix $Ben_{n,k}$ is calculated by (3). This way, any feasible solution S fulfilling objective (7a) and constraint (7b) also fulfills objective (5). The constraint (7c) of the MDS problem shows that if a node v_i is not selected, there is at least one neighbor of v_i selected in the solution s. Similarly, the benefit of app user $u \in U_i$ can be obtained as $b_u \geq 1$. This way, any feasible solution S fulfilling the constraint (7c) also fulfills the constraints (1) (2) (3) and (4). Therefore, the EDC problem is reducible from MDS and it is $\mathcal{NP}$-complete. □

4. Edge data caching strategy

We first propose the optimal model solved by Lexicographic Goal Programming[1] technique, then provide an efficient approximation algorithm with the analysis of its approximation ratio.

4.1. Optimal model

The EDC problem can be modeled as a constrained optimization problem (COP). One of the two optimization objectives can be prioritized over the other with the Lexicographic Goal Programming technique, depending on the app vendor's preference.

Given $G = (V, E)$, where $V = \{v_1, .., v_n, \}$ and $E = \{e_1, \ldots, e_m\}$, there is a set of variables $R = \{r_1, .., r_n\}$, where $r_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\}$, r_i being 1 if the data replica is cached on the ith node, or 0 otherwise. The constraints for the COP model are:

$$b_u = \max\{r_i * b_{u,i}\}, \forall u \in \{1, \ldots, k\}, \forall i \in \{1, \ldots, n\} \tag{8}$$

$$1 \leq b_u \leq 2, \forall u \in \{1, \ldots, k\} \tag{9}$$

Constraint family (8) is calculated by (4), which guarantees that all the app users can retrieve data from the nearest edge server. Constraint family (9) enforces the latency constraint to ensure that every app user can retrieve the data from an edge server within one hop, which means the data is retrieved from the app user's local edge server via zero hops or neighbor edge server via one hop.

To satisfy constraint families (8) and (9), there might be multiple solutions. For example, two possible data caching solutions in Fig. 2(a) and Fig. 2(b) are $R_1 = \{0, 1, 1, 1, 0, 0\}$, which caches the data on v_2, v_3, and v_4, and $R_2 = \{1, 0, 0, 0, 0, 1\}$, which caches the data on v_1 and v_6. Both R_1 and R_2 are feasible with consideration of (8) and (9). However, the data caching cost of R_2 is less than that of R_1, where $cost(R_1) = 3$ and $cost(R_2) = 2$. To minimize the data caching cost, the below objective in the COP model is included to capture the app vendor's first optimization objective:

$$\min \sum_{i=1}^{n} r_i \tag{10}$$

The app vendor's second optimization objective also needs to be modeled in the COP. Let us assume two solutions as demonstrated in Fig. 2(b) and Fig. 2(c), $R_2 = \{1, 0, 0, 0, 0, 1\}$, which caches the data on v_1 and v_6, and $R_3 = \{0, 1, 0, 1, 0, 0\}$, which caches the data on v_2 and v_4, both fulfilling the latency constraint and achieving the app vendor's first optimization objective to minimize the data caching cost. However, compared with v_1 and v_6, v_2 and v_4 cover more app users, i.e., 39 versus 13 in total. Thus, R_3 allows more app users to retrieve the data from their local edge servers. Thus, from the app vendor's perspective, R_3 produces more caching benefits (i.e., lower retrieval latency) than R_2 at the same data caching cost. The below objective function that maximizes the data caching benefits of all app users based on (4) is included in the COP model to capture the app vendor's second optimization objective:

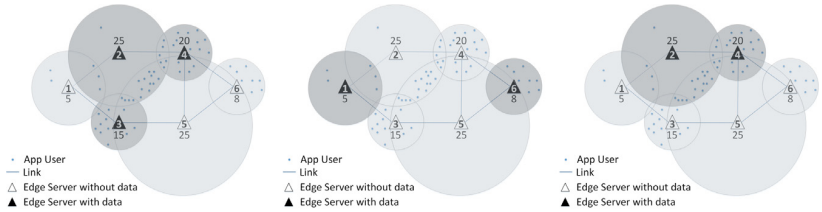
$$\max \sum_{u=1}^{k} b_u \tag{11}$$

The COP above can be solved with Integer Programming problem solvers, such as Gurobi[2] and IBM CPLEX Optimizer.[3] Here we name this optimal solution as IPEDC.

[1] https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.9.0/ilog.odms.cplex.help/CPLEX/UsrMan/topics/multiobj/multiobj_intro.html.

[2] http://www.gurobi.com/.

[3] https://www.ibm.com/analytics/cplex-optimizer.



(a) Strategy R_1 (b) Strategy R_2 (c) Strategy R_3

Fig. 2. Example data caching strategies.

4.2. Approximation algorithm

As mentioned in Section 3.3, the EDC problem is $\mathcal{NP}$-complete. It is intractable to find optimal solutions to large-scale EDC problems. In [7], we proposed a simple greedy algorithm, namely LGEDC. However, there is a significant performance gap between IPEDC and LGEDC. Thus, we develop a new approximation algorithm named AEDC to achieve higher performance with lower computation overhead than LGEDC.

Given $V = \{v_1, \ldots, v_n\}$ and $U = \{u_1, \ldots, u_m\}$, AEDC implements an iterative process for app vendors to hire edge servers to cache data replicas. The pseudo code is presented in Algorithm 1.

The algorithm starts with the initialization in Lines 1–4. To calculate the benefits of the solution provided by this algorithm, DCU is introduced to present the set of app users already covered by the solution, while CU means the set of app users that can access data via one hop. As edge servers can communicate with their neighbor edge servers, for each server $v_i \in V$, cu_i can be presented as the set of app users in the coverage of edge server v_i and their neighbor edge servers (Line 5).

In Algorithm 1, Δcu_i and Δdcu_i are used to select edge servers to cache data replicas, where Δcu_i is the number of users in cu_i but not in CU and Δdcu_i presents the number of users in dcu_i but not in DCU. For each iteration, Δcu_i and Δdcu_i are updated for each edge server $v_i \in V$. Then edge server v, which has the maximum value of Δcu, would be included into the solution set S to maximize the number of newly covered users. If there are more than one edge servers with that maximum value, AEDC selects the one with the maximum value of Δdcu to earn more benefits. This process iterates until all the app users are covered by the set of selected edge servers.

Fig. 3 presents the system state at different moments during the AEDC process for the EDC example in Fig. 1. Fig. 3(a) illustrates the initial state. Then, edge server v_5 is selected to cache a data replica because v_5 can serve the most new app users (54 users) via one hop where $\Delta CU = \{42, 41, 38, 50, 54, 46\}$ in Fig. 3(b). After the first iteration, there are 6 users not covered by the solution set $S = \{v_5\}$, and the algorithm continues the iterative process to choose another edge server. In the second iteration, both Δcu_1 and Δcu_2 are the maximal value while $\Delta CU = \{6, 6, 5, 5, 0, 0\}$. In this case, edge server v_1 is selected in Fig. 3(c), because $\Delta dcu_1 = 5 > \Delta dcu_2 = 4$. Thus, the solution of AEDC is $S = \{v_5, v_1\}$, as all users have been covered by selected edge servers. Considering the optimal solution $R_3 = \{0, 1, 0, 1, 0, 0\}$ described in Section 4.1, $Cost_{IPEDC} = Cost_{AEDC} = 2$ while $Benefits_{IPEDC} = 99 > Benefits_{AEDC} = 90$.

As the IPEDC approach is implemented based on the Integer Programming technique, IPEDC selects the optimal solution among all the possible solutions that have a total of $C_n^1 + C_n^2 + \cdots + C_n^n = 2^n$ possible results. Moreover, we prove that the COP of EDC is NP-complete in Section 3.3, thus IPEDC cannot find the optimal solution within polynomial time. In the AEDC algorithm, it can be calculated that the worst case of computational complexity is $O(n^2)$. This means that AEDC can reduce a large amount of execution time to find the solution of EDC problem, compared with IPEDC.

Algorithm 1: AEDC Algorithm.

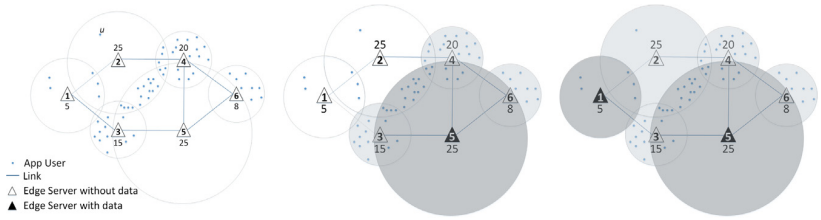
```
1:  Initialization:
2:  CU, DCU, S ← ∅, benefits, cost = 0
3:  for each server v_i do
4:      dcu_i ← U_i, cu_i ← U_i
5:  end for
6:  End of initialization
7:  for each v_i ∈ V do
8:      for each neighbor v_j of v_i do
9:          cu_i ← cu_i ∪ dcu_j
10:     end for
11: end for
12: repeat
13:     for each v_i ∈ V do
14:         Δcu_i = |cu_i ∩ ¬CU|
15:         Δdcu_i = |dcu_i ∩ ¬DCU|
16:     end for
17:     v ← v_0
18:     for each v_i ∈ V do
19:         if Δcu_{v_i} > Δcu_v or
            Δcu_{v_i} = Δcu_v and Δdcu_{v_i} > Δdcu_v then
20:             v ← v_i
21:         end if
22:     end for
23:     S ← S ∪ {v}
24:     CU ← CU ∪ cu_i
25:     DCU ← DCU ∪ dcu_i
26: until CU = U
27: return S
```

Now, we prove the approximation ratio of AEDC, where it is the ratio of the cost incurred by AEDC and that incurred by IPEDC in the worst case.

Theorem 2. *The approximation ratio of AEDC is $\ln\Delta + 1$, where Δ denotes the maximum number of app users that can be covered by any edge server $v \in V$ within 1 hop.*

Proof. As mentioned in Section 2, the number of data replicas is used to measure the data caching cost. For each edge server selected to cache a data replica, the cost is 1. Here we implement an amortized strategy to analyze the approximation ratio. This way, we distribute the cost 1 to newly covered users equally instead of the selected edge server. Take Fig. 3(b) as an example, the cost of each user incurred by selecting v_5 is $\frac{1}{54}$.

Based on constraint (9), all the app users should be covered by the data caching strategy. Let S_{opt} denote the optimal solution found by IPEDC. In this case, we can divide the graph G into $|S_{opt}|$ stars. Each star contains 1 edge server as the center of the star and the newly covered users as its leaves.



(a) $S = \varnothing$ (b) $S = \{v_5\}$ (c) $S = \{v_5, v_1\}$

Fig. 3. Example system state at different moments during AEDC process.

Denote $newU(v)$ as the number of newly covered app users by adding edge server v into the solution. For each edge server $v \in S_{opt}$, the corresponding star has $newU(v)$ leaves. Based on the heuristic logic of AEDC, the distributed cost is at most $\frac{1}{newU(v)}$. Otherwise, AEDC would choose another edge server. After assigning the distributed cost to newly covered app users, those app users would not produce any more costs. In the worst case, no two app users in the same star are covered together. Thus, the total distributed cost of a star is at most:

$$\frac{1}{newU(v)} + \frac{1}{newU(v)-1} + \cdots + \frac{1}{2} + \frac{1}{1}$$

Based on the harmonic series, we can obtain that the total distributed cost of a star is at most $ln(newU(v)) + 1$. Thus, the total cost produced by AEDC is at most $|S_{opt}| \cdot (ln(newU(v)) + 1)$. Moreover, for each $v \in V$, $newU(v)$ is always less than or equal to Δ. Thus, the approximation ratio of AEDC is:

$$ratio \leq \frac{|S_{opt}| \cdot (ln\Delta + 1)}{|S_{opt}|} = ln\Delta + 1 \quad \square \tag{12}$$

5. Experimental evaluation

We experimentally evaluate IPEDC and AEDC on a widely-used real-world data set and a synthetic data set, and compare their performance against four representative approaches.

5.1. Comparison approaches

In these experiments, we evaluate and compare the performance of IPEDC and AEDC against four comparison approaches, namely *LGEDC*[7], *DIP*,*liu2019data*, *Random* and *Greedy-Covered-Users*:

- *LGEDC*: This algorithm keeps selecting the edge server with the most links until it satisfies the latency constraint (2).
- *DIP* [15]: This approach tries to minimize the app vendor's revenue, combining caching cost and latency reduction, by caching data on edge servers without leveraging the collaboration between edge servers. Its parameter settings in the experiments are the same as [15], i.e., [5, 25] for the unit cost of cache and [0.5, 2] for the unit cost of latency (same as unit benefit). Accordingly, the range of the ratio between them is [2.5, 50]. In the EDC problem, all users must be covered. To pursue this goal, the ratio is fixed at 2.5 for DIP in the experiments to cover as many users as possible.
- *Random*: This algorithm keeps selecting the edge server randomly until it satisfies the latency constraint (2).
- *Greedy-Covered-Users*(GU): This algorithm keeps selecting the edge server with the most app users until it satisfies the latency constraint (2).

5.2. Experimental settings

Data Sets: There are two sets of experiments. The first set is conducted the public real-world Edge User Allocation (EUA) data set[4][4] with 128 edge servers and 816 app users in the Melbourne CBD area. The second set is conducted on a synthetic data set that simulates more general EDC scenarios. In the experiments on the synthesized data set, a specific number of edge servers are randomly distributed within a particular area with app users also generated randomly. In both sets of experiments, edges are randomly generated to connect the edge servers according to the edge density to ensure that the graph is connected.

Parameter Settings: To comprehensively analyze IPEDC and AEDC, we vary two parameters in the experiments to simulate different EDC scenarios, as presented in Table 2. This way, we can also analyze how the changes in the parameters impact the performance of our approaches. Each experiment is repeated 100 times every time we change a parameter, and the results are averaged:

- The total number of edge servers ($n = |V|$). In experiment Set #1 and Set #2.1, this number varies from 10 to 50 in steps of 10.
- Edge density ($d = |E|/|V|$). In experiment Set #2.2, this number varies from 1 to 3 in steps of 0.4.

Performance Metrics: In the experiments, we use four metrics to evaluate the effectiveness and efficiency of all the approaches:

1. Data Caching Cost *cost*, the lower the better;
2. Data Caching Benefit *benefit*, the higher the better;
3. Benefit per Data Replica *bpr*, the higher the better; and
4. Computation Overhead *time*, the lower the better.

To stabilize the impact of the number of app users, we always generate 100 app users in experiment Set #2.

5.3. Experimental results

Figs. 4–6 show the results of the experiments Set #1, #2.1 and #2.2, respectively.

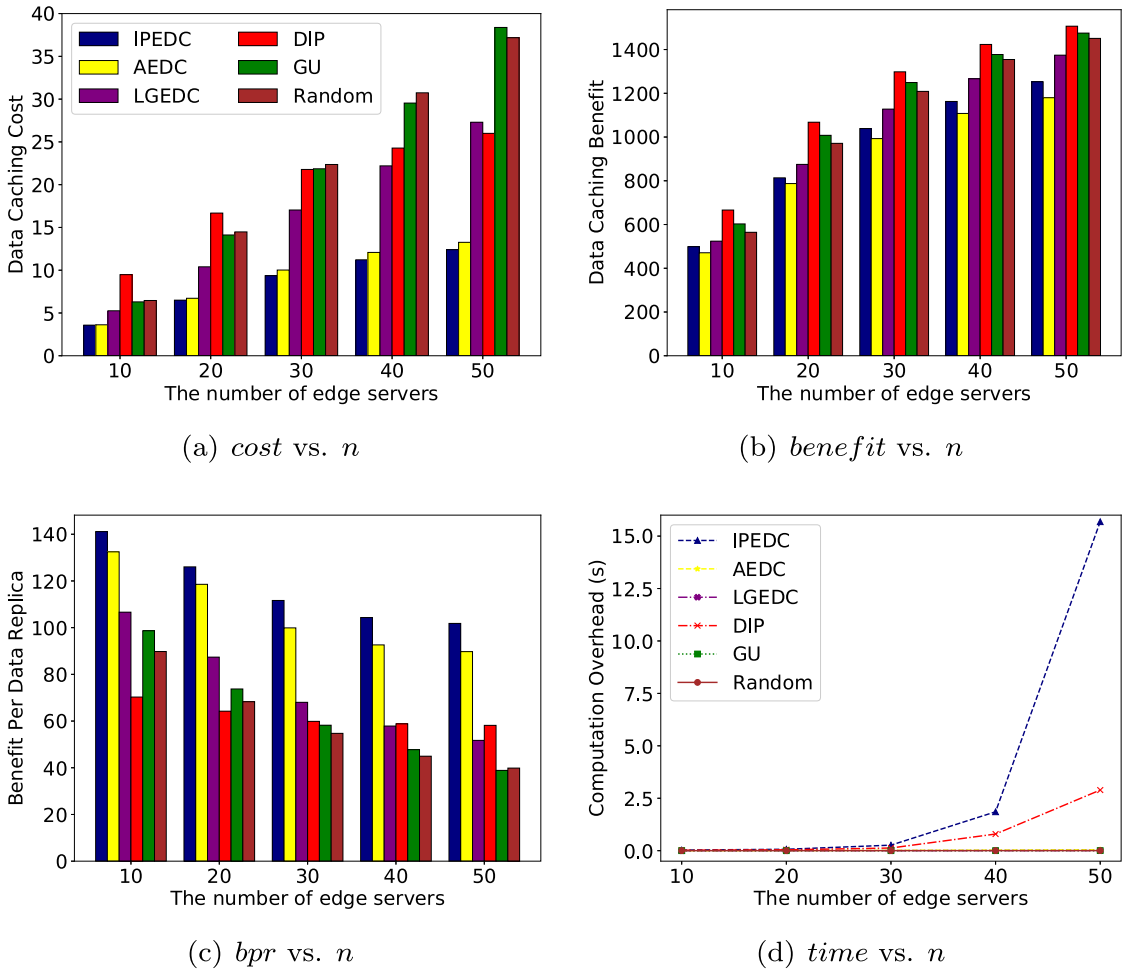
5.3.1. Effectiveness

Fig. 4 shows the results of experiment Set #1. Overall, of all the six approaches, **IPEDC achieves the highest benefit per data replica at the lowest data caching cost, while AEDC is the second lowest in cost with the second highest in benefit per data replica**. Fig. 4(b) shows that AEDC and IPEDC achieve lower data caching benefits than those four comparison approaches. With the priority to minimize the data caching cost, if an app user can retrieve data from edge servers via one hop, there is no need to do so via zero hops. Thus, IPEDC will aim for a solution

[4] https://github.com/swinedge/eua-dataset.

Table 2
Parameter settings.

	Number of edge servers	Edge density	Data set
Set #1	10, 20, 30, 40, 50	1	Real-world
Set #2.1	10, 20, 30, 40, 50	1	Synthetic
Set #2.2	30	1, 1.4, 1.8, 2.2, 2.6, 3	Synthetic



(a) *cost* vs. n (b) *benefit* vs. n

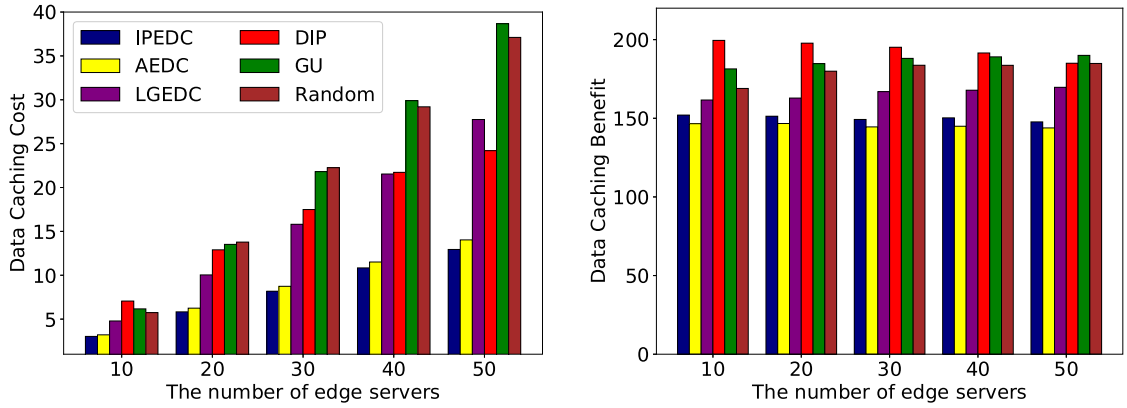
(c) *bpr* vs. n (d) *time* vs. n

Fig. 4. Experiment set #1.

that barely fulfills (9), i.e., a solution just good enough to allow as many app users as possible to retrieve data from edge servers via one hop.

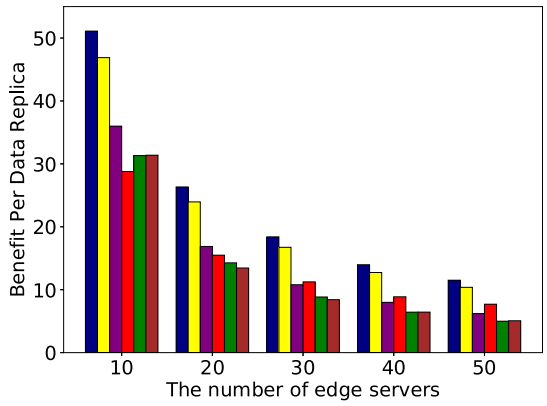
Fig. 4(a) shows that **the average data caching costs achieved by IPEDC and AEDC are much lower than the other four approaches** across all five cases, i.e., 8.61 (IPEDC) and 9.14 (AEDC) versus 16.44 (LGEDC), 19.64 (DIP), 22.04 (GU) and 22.24 (Random). The average advantage of IPEDC is 5.80% against AEDC, 47.63% against LGEDC, 56.16% against DIP, 60.93% against GU, and 61.29% against Random. Fig. 4(a) also shows that, as the number of edge servers increases from 10 to 50, the data caching cost achieved by AEDC increases from 3.62 replicas to 13.26 replicas on average, similar to IPEDC (3.58 to 12.4) but much slower than LGEDC (5.26 to 27.3), DIP (9.48 to 26.0), GU (6.34 to 38.38) and Random (6.46 to 37.18). Fig. 4(b) shows that the increase in the number of edge servers will increase the data caching benefits achieved by all six approaches, from 498.68 to 1252.82 for IPEDC, 470.90 to 1179.74 for AEDC, 523.98 to 1374.32 for LGEDC, 666.24 to 1506.76 for DIP, 602.82 to 1475.24 for GU and 564.48 to 1451.04 for Random. Fig. 4(c) shows **the significant advantages of IPEDC and AEDC over the other approaches** in achieving cost-effective data caching strategies. IPEDC has the best performance, which averagely outperforms AEDC by 9.70%, LGEDC by 57.39%, DIP by 87.88% GU by 84.38% and Random by 96.57%.

Fig. 5 depicts the results of experiment Set # 2.1. Overall, **IPEDC again achieves the highest data caching benefit per replica at the lowest data caching cost**, following by AEDC. The advantages of IPEDC and AEDC over the other four approaches are significant. In this set of experiments, the edge servers are set up in a similar way as in Set #1. Therefore, the results shown in Fig. 5(a) are similar to those shown in Fig. 4. However, Fig. 5(b) shows that the **data caching benefit does not increase** with the increase in the number of edge servers. The reason is that, unlike experiment Set #1, the number of app users in experiment Set #2.1 does not increase. Thus, the data caching benefit does not increase accumulatively as in Fig. 4(b). This is also the same reason for the rapid decrease in the benefit per data replica demonstrated in Fig. 5(c).

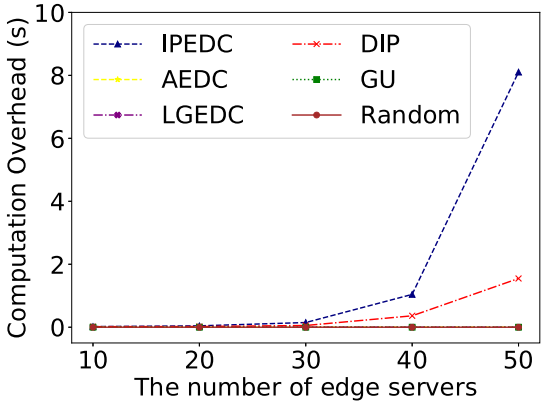


(a) *cost* vs. n

(b) *benefit* vs. n



(c) *bpr* vs. n



(d) *time* vs. n

Fig. 5. Experiment set #2.1.

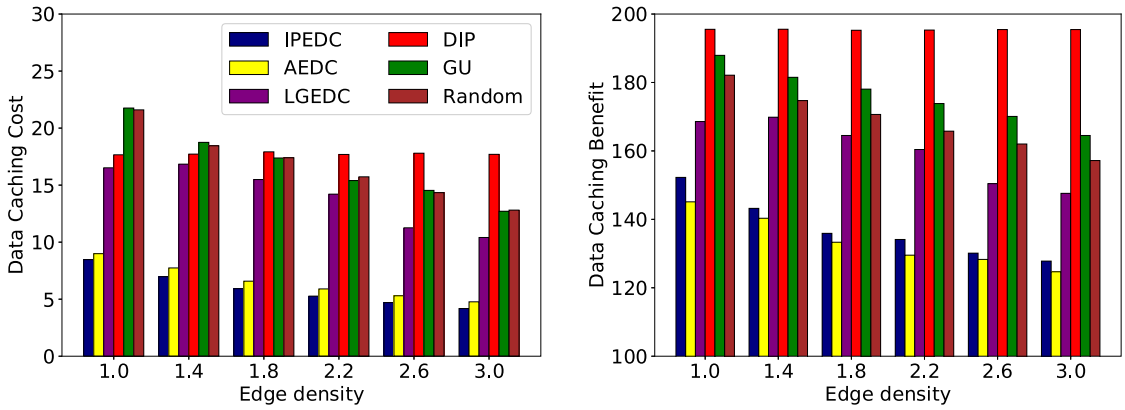
Fig. 6 shows the results in experiment Set 2.2 where the edge density varies. In terms of the average data caching cost and benefit per data replica, **IPEDC and AEDC outperform the other four approaches** with large margins, and IPEDC still has the best performance. The advantage of IPEDC is 9.62% against AEDC, 58.07% against LGEDC, 66.65% against DIP, 64.68% against GU and 64.59% against Random on average in data caching cost, while 13.38% against AEDC, 94.67% against LGEDC, 120.43% against DIP, 116.73% against GU and 125.71% against Random on average in benefit per data replica.

Interestingly, Fig. 6 shows that **the edge density impacts the approaches in a very different way** from the number of edge servers. Fig. 6(a) shows that as the edge density increases from 1.0 to 3.0, the data caching costs achieved by IPEDC and AEDC decrease from 8.47 to 4.18 and from 8.99 to 4.77 respectively. After investigating the results, we find that the increase in the edge density allows each edge server to link to more edge servers. This increases the app users' chances of retrieving data from edge servers via one hop. IPEDC does not need to cache as many data replicas to ensure that all app users are served by edge servers within one hop. As a result, the average data caching cost decreases. For the same reason, the data caching benefit decreases, as demonstrated in Fig. 6(b). The increase in the connectivity between edge servers also allows more app users to be able to retrieve data via one hop. As a result, the benefit per data replica increases, as demonstrated in Fig. 6(c), from 18.18 to 31.01 for IPEDC, from 16.35 to 26.97 for AEDC, from 10.41 to 16.15 for LGEDC, from 11.20 to 11.15 for DIP, from 8.89 to 13.67 for GU and from 8.60 to 13.25 for Random. Since DIP focuses on maximizing the benefits with consideration of caching cost, the total benefits achieved by DIP are only changed slightly when the maximum benefits are fixed in Set #2.

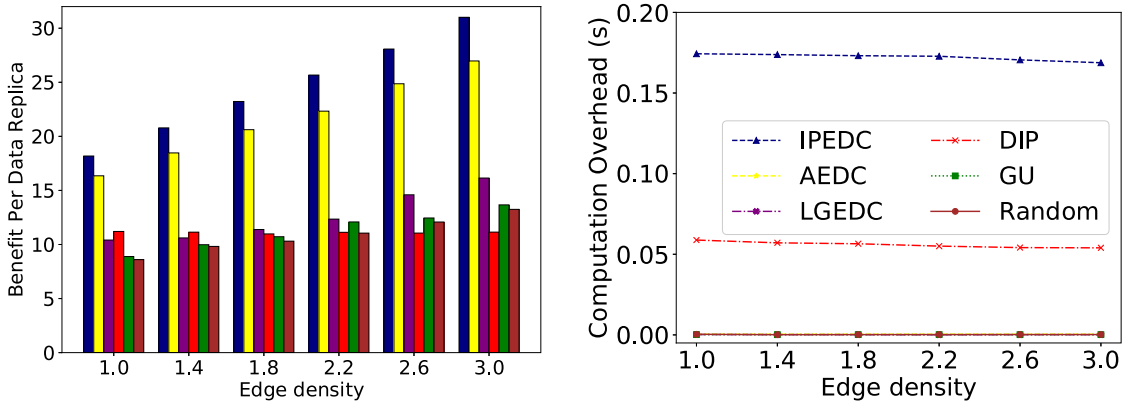
Overall, our **IPEDC and AEDC outperform LGEDC, DIP, GU and Random significantly and consistently** in formulating cost-effective data caching strategies. Overall, AEDC can achieve about 90% of IPEDC's performance in minimizing data caching cost and benefits per replica across all the experiments. Both IPEDC and AEDC are particularly effective in EDC scenarios where edge servers are highly connected.

5.3.2. Efficiency

Figs. 4(d), 5(d) and 6(d) present the average computation overheads of the six approaches in finding a solution to the EDC problem. We can see in Figs. 4(d) and 5(d) that the **computation overhead of IPEDC increases rapidly** when the number of edge servers increases. When there are 50 edge servers to consider, IPEDC takes more than 15 s to find the optimal solution, as shown in Fig. 4(d). Excessive computation overheads are inevitable when IPEDC is looking for the optimal solution to this NP-complete EDC problem. Thus, IPEDC is **suitable for solving EDC problems in small sizes**. To solve large-scale EDC problems, **heuristics-based approaches are more practical**, e.g., AEDC, LGEDC or GU. The



(a) *cost* vs. d (b) *benefit* vs. d



(c) *bpr* vs. d (d) *time* vs. d

Fig. 6. Experiment set #2.2.

results in Fig. 6(d) indicate that both IPEDC and AEDC are capable for handling dense graphs built based on edge servers that are highly connected. In conclusion, IPEDC can be used to solve EDC problems in small sizes while AEDC handles the large-scale ones.

5.4. Threats to validity

Threat to construct validity. The main threat to construct validity is the four comparison approaches. Due to the novelty of this edge data caching problem, we choose the greedy approach proposed in [7], the optimal benefit-based approach proposed in [15] and two basic baseline approaches to compare with our approaches in the experiments. To minimize the threat of comprehensive evaluation, two parameters are varied in the experiments to simulate different EDC scenarios. This way, we could not only evaluate IPEDC and AEDC with four comparison algorithms but also present the impacts of varying parameters on those algorithms.

Threat to external validity. The main threat here is whether IPEDC and AEDC are also suitable in other edge computing scenarios. To address this, we formulate the approaches and measure the performance in a more generic way: evaluating the effectiveness by using the number of data replicas and the number of hops for cost and benefit. Moreover, the two data sets used to conduct the experiments, including a real-world one and a synthetic one. Thus, the representativeness and comprehensiveness of the evaluation are ensured, and this threat is reduced.

Threat to conclusion validity. The lack of statistical tests, e.g., chi-square tests, is the major threat to conclusion validity in our paper. To compensate this threat, we have conducted comprehensive and intensive experiments to cover various scenarios in different size and complexity. Every time a parameter changes, we repeat the experiment for 100 times and calculate the averaged results. This led to a large number of test cases, which tend to result in a small p-value in the chi-square tests and lower the practical significance of the test results [16]. For example, in experiment Set #2, there were a total of 1100 runs. This number is not even close to the number of observation samples that concern Lin et al. in [16]. Thus, the threat to the conclusion validity due to the lack of statistical tests might be high but not significant.

5.5. Real-world applications

User mobility is an important characteristic of the real-world edge computing environment. However, the structure of an edge server graph does not change when the users move in the area. It is determined by how the edge servers in the area are physically linked. Moreover, for its high efficiency, AEDC can solve the EDC problem quickly in different time slots, similar to the approaches proposed in [9,14]. In each time slot, edge servers need to update the information including users within its coverage area and their requests. Specifically, when a user no longer needs a piece of

data or has moved out of the current edge server's coverage area, this user can be removed from the current edge server's user set U_i. In the meantime, new users can be added into U_i. This way, the context can be rapidly updated at the start of each time slot. Then, the AEDC algorithm can be executed in each time slot to formulate the corresponding data caching strategies. Given its high efficiency as demonstrated in Figs. 4(d), 5(d) and 6(d), AEDC can be executed periodically or on-demand to adapt to user mobility in real-world EDC scenarios.

6. Related work

Data caching have been extensively investigated in the fields of conventional distributed computing and cloud computing environments. With the popularity of edge computing, data caching in the edge computing environment is obtaining attention from researchers recently.

6.1. Conventional distributed data Caching

In the last few decades, there are many data caching problems investigated in conventional distributed computing environments, including web caching [17], content-centric networking [18], content delivery network [19], etc. Banerjee et al. [20] developed a content placement strategy for information-centric network based on data popularity, namely Greedy Caching. With popular contents cached in the network, the Greedy Caching approach considered the cache miss rate at the edge to decide what contents would be cached on the core server. In [21], the authors formulated two caching strategies for data publish–subscribe systems, including eviction-based caching and time-to-live-based caching to address the space and time issues, respectively. The authors of [22] focused on balancing the trade-off between latency and cost in the content-centric network. They addressed this issue with a holistic model for provisioning the storage capability based on the network performance and the provisioning cost.

6.2. Cloud data Caching

In the cloud computing environment, a critical problem of data caching is how to utilize cache space efficiently on cloud hosts and mobile devices.

Arteaga et al. [23] proposed CloudCache, a method for managing cache, to fulfill the caching requirement of the workload and minimize cache wear-out. In [24], the authors presented how to use segment access-aware dynamic semantic cache in the cloud computing environment for relational databases. A cache access algorithm was introduced to consider cache exact hit, cache extended hit, cache partial hit and cache miss. The authors of [25] explored the cache design space for embedded processors with evolutionary techniques for mobile and thin client processors in the cloud computing environment. A heuristic and evolutionary method was presented to generate a near-optimal cache space design for enhancing service quality. In [26], the authors formulated a benefit maximization problem and created a cache replacement approach based on spatio-temporal traffic requirements. They also introduced a content clustering method for collecting popular data and clustering similar contents.

6.3. Edge data Caching

As an extension of cloud computing, edge computing distributes both computing capacities and storage resources from cloud server to edge servers [27]. With the deployment of edge servers, the problems of computation offloading and data caching occurs. The computation offloading problems have been extensively investigated from different perspectives, including edge servers' energy efficiency [28] and offloading cost [29].

Recently, there are some researchers starting to investigate the data caching problems in edge computing. As mentioned in Section 2, the data caching strategies from conventional distributed computing and cloud computing cannot be directly applied in edge computing. Thus, those researches introduced new ideas and approaches. Cao et al. proposed an optimal auction mechanism with the consideration of the costs produced during delivery and retrieval. The authors of [30] provided a caching system, namely Agar, from the erasure-coded perspective. Agar was a dynamic programming algorithm which could cache data chunks optimally with consideration of data popularity and network latency.

Instead of improving internal cache utility on edge servers, some researchers started to investigate how to combine the advantages of both internal caches and external caches. Zhang et al. [31] integrated in-network caching and edge caching to ensure the latency requirements of time-sensitive transmissions over the 5G network. The authors of [32] introduced a new edge caching architecture with improved resource utility by using smart vehicles as external edge caches.

The above studies mostly focus on cost savings. Data latency is also an important issue in edge data caching problems. Drolia et al. [33] proposed an edge caching system, namely Cachier, to minimize the data retrieval latency. They implemented a coordinating mechanism to balance the loads between the cloud server and edge servers dynamically. Liu et al. [15] studied the data caching problem in the edge computing environment with the aim to maximize the app vendor's revenue based on caching cost and latency. This approach was implemented in our experiments as DIP for comparison. However, those work ignored the collaboration between edge servers, as well as the benefit produced by the reduction in user' service latency.

Edge computing inherits the pay-as-you-go price model from cloud computing. Thus, the cost incurred for app vendors is critical to the success of edge computing because they are the main customers in the edge computing environment. However, all the above work tackles the data caching problem from network providers' or app users' perspectives, our work solves the Edge Data Caching (EDC) problem from the app vendor' perspective in the edge computing environment. We also realistically and innovatively solve the EDC problem in a generic manner to minimize the data caching cost and maximize the data caching benefit with the server coverage constraint and the server adjacency constraint.

7. Conclusion

In this paper, we formulated the new Edge Data Caching (EDC) problem as a constrained optimization problem based on the graph from the app vendor's perspective. The optimal solution of the EDC problem is to find a solution that minimizes the data caching cost and maximizes the data caching benefit. We proved that the EDC problem is $\mathcal{NP}$-complete. Then we proposed an optimal solution IPEDC solved by Integer Programming, and an approximation algorithm AEDC within a provable bound. We conducted extensive experiments based on a real-world data set and a synthetic data set to evaluate our approaches. The results demonstrate that both IPEDC and AEDC significantly outperform all other four baseline approaches in formulating cost-effective EDC solutions, while AEDC solves large-scale EDC problems efficiently.

This research has established the foundation for the EDC problem and opened up a number of research directions. In the future, we will investigate multiple data caching scenarios, partitionable data caching scenarios, users' dynamic participation and security policy. Those will allow our approaches to accommodate more sophisticated EDC scenarios.

CRediT authorship contribution statement

Xiaoyu Xia: Conceptualization, Methodology, Software, Investigation, Writing - original draft. **Feifei Chen:** Conceptualization, Methodology, Writing - review & editing, Supervision. **Qiang He:** Conceptualization, Methodology, Writing - review & editing, Supervision, Funding acquisition. **Guangming Cui:** Conceptualization, Methodology, Software. **Phu Lai:** Conceptualization, Methodology, Software. **Mohamed Abdelrazek:** Conceptualization, Writing - review & editing, Supervision, Funding acquisition. **John Grundy:** Conceptualization, Writing - review & editing, Supervision, Funding acquisition. **Hai Jin:** Conceptualization, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is partially funded by Australian Research Council Projects No. DP170101932, DP180100212 and Laureate Fellowship, Australia FL190100035.

References

[1] Afif Osseiran, Volker Braun, Taoka Hidekazu, Patrick Marsch, Hans Schotten, Hugo Tullberg, Mikko A Uusitalo, Malte Schellman, The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions, in: IEEE 77th Vehicular Technology Conference, VTC2013-Spring, 2013, pp. 1–5.

[2] Anthony D. Josep, Randy Katz, Andy Konwinski, Lee Gunho, David Patterson, Ariel RabKin, A view of cloud computing, Commun. ACM 53 (4) (2010).

[3] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.

[4] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, Yun Yang, Optimal edge user allocation in edge computing with variable sized vector bin packing, in: International Conference on Service-Oriented Computing, 2018, pp. 230–245.

[5] Tuyen X. Tran, Mohammad-Parsa Hosseini, Dario Pompili, Mobile edge computing: Recent efforts and five key research directions, IEEE COMSOC MMTC Commun.-Frontiers 12 (4) (2017) 29–33.

[6] Qiang He, Guangming Cui, Xuyun Zhang, Feifei Chen, Shuiguang Deng, Hai Jin, Yanhui Li, Yun Yang, A game-theoretical approach for user allocation in edge computing environment, IEEE Trans. Parallel Distrib. Syst. (2019).

[7] Xiaoyu Xia, Feifei Chen, Qiang He, Guangming Cui, Phu Lai, Mohamed Abdelrazek, John Grundy, Hai Jin, Graph-based optimal data caching in edge computing, in: International Conference on Service-Oriented Computing, Springer, 2019, pp. 477–493.

[8] VNI Cisco Mobile, Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper, 2017.

[9] Lixing Chen, Sheng Zhou, Jie Xu, Computation peer offloading for energy-constrained mobile edge computing in small-cell networks, IEEE/ACM Trans. Netw. 26 (4) (2018) 1619–1632.

[10] Min Chen, Yixue Hao, Kai Lin, Zhiyong Yuan, Long Hu, Label-less learning for traffic control in an edge network, IEEE Netw. 32 (6) (2018) 8–14.

[11] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, Wenbo Wang, A survey on mobile edge networks: Convergence of computing, caching and communications, IEEE Access 5 (2017) 6757–6779.

[12] Alexandru Tatar, Marcelo Dias De Amorim, Serge Fdida, Panayotis Antoniadis, A survey on predicting the popularity of web content, J. Internet Serv. Appl. 5 (1) (2014) 1–20.

[13] Xu Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2014) 974–983.

[14] Jie Xu, Lixing Chen, Pan Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 207–215.

[15] Ying Liu, Qiang He, Dequan Zheng, Mingwei Zhang, Feifei Chen, Bin Zhang, Data caching optimization in the edge computing environment, in: 2019 IEEE International Conference on Web Services, ICWS, IEEE, 2019, pp. 99–106.

[16] Mingfeng Lin, Henry C. Lucas Jr., Galit Shmueli, Research commentary—too big to fail: large samples and the p-value problem, Inf. Syst. Res. 24 (4) (2013) 906–917.

[17] Stefan Podlipnig, Laszlo Böszörmenyi, A survey of web cache replacement strategies, ACM Comput. Surv. 35 (4) (2003) 374–398.

[18] Yonggong Wang, Zhenyu Li, Gareth Tyson, Steve Uhlig, Gaogang Xie, Optimal cache allocation for content-centric networking, in: 21st IEEE International Conference on Network Protocols, ICNP, 2013, pp. 1–10.

[19] Daniel S. Berger, Ramesh K. Sitaraman, Mor Harchol-Balter, AdaptSize: Orchestrating the hot object memory cache in a content delivery network, in: 14th USENIX Symposium on Networked Systems Design and Implementation, 2017, pp. 483–498.

[20] Bitan Banerjee, Adita Kulkarni, Anand Seetharam, Greedy caching: An optimized content placement strategy for information-centric networks, Comput. Netw. 140 (2018) 78–91.

[21] Md Yusuf Sarwar Uddin, Nalini Venkatasubramanian, Edge caching for enriched notifications delivery in big active data, in: 38th IEEE International Conference on Distributed Computing Systems, ICDCS, 2018, pp. 696–705.

[22] Yanhua Li, Haiyong Xie, Yonggang Wen, Zhi-Li Zhang, Coordinating in-network caching in content-centric networks: Model and analysis, in: 33rd IEEE International Conference on Distributed Computing Systems, ICDCS, 2013, pp. 62–72.

[23] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, Ming Zhao, CloudCache: On-demand flash cache management for cloud computing, in: 14th USENIX Conference on File and Storage Technologies, FAST, 2016, pp. 355–369.

[24] Kun Ma, Bo Yang, Zhe Yang, Ziqiang Yu, Segment access-aware dynamic semantic cache in cloud computing environment, J. Parallel Distrib. Comput. 110 (2017) 42–51.

[25] Abdel-Hameed A Badawy, Gabriel Yessin, Vikram Narayana, David Mayhew, Tarek El-Ghazawi, Optimizing thin client caches for mobile cloud computing: Design space exploration using genetic algorithms, Concurr. Comput.: Pract. Exper. 29 (11) (2017) e4048.

[26] Syed Tamoor-ul Hassan, Sumudu Samarakoon, Mehdi Bennis, Matti Latva-Aho, Choong Seon Hong, Learning-based caching in cloud-aided wireless networks, IEEE Commun. Lett. 22 (1) (2018) 137–140.

[27] Marcelo Yannuzzi, Frank van Lingen, Anuj Jain, Oriol Lluch Parellada, Manel Mendoza Flores, David Carrera, Juan Luis Pérez, Diego Montero, Pablo Chacin, Angelo Corsaro, et al., A new era for cities with fog computing, IEEE Internet Comput. 21 (2) (2017) 54–67.

[28] Feng Wang, Jie Xu, Xin Wang, Shuguang Cui, Joint offloading and computing optimization in wireless powered mobile-edge computing systems, IEEE Trans. Wireless Commun. 17 (3) (2018) 1784–1797.

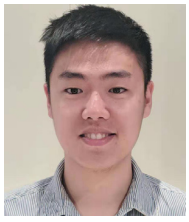
[29] Hong Yao, Changmin Bai, Muzhou Xiong, Deze Zeng, Zhangjie Fu, Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing, Concurr. Comput.: Pract. Exper. 29 (16) (2017) e3975.

[30] Raluca Halalai, Pascal Felber, Anne-Marie Kermarrec, François Taïani, Agar: A caching system for erasure-coded data, in: 37th IEEE International Conference OnDistributed Computing Systems, ICDCS, 2017, pp. 23–33.

[31] Xi Zhang, Qixuan Zhu, Hierarchical caching for statistical qos guaranteed multimedia transmissions over 5g edge computing mobile wireless networks, IEEE Wirel. Commun. 25 (3) (2018) 12–20.

[32] Ke Zhang, Supeng Leng, Yejun He, Sabita Maharjan, Yan Zhang, Cooperative content caching in 5g networks with mobile edge computing, IEEE Wirel. Commun. 25 (3) (2018) 80–87.

[33] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, Priya Narasimhan, Cachier: Edge-caching for recognition applications, in: 37th IEEE International Conference OnDistributed Computing Systems, ICDCS, 2017, pp. 276–286.

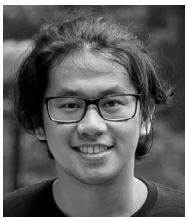


Xiaoyu Xia received his Master degree from The University of Melbourne, Australia in 2015. He is a Ph.D. candidate at Deakin University. His research interests include edge computing, service computing and software engineering.

Feifei Chen received her Ph.D. degree from Swinburne University of Technology, Australia in 2015. She is a lecturer at Deakin University. Her research interests include software engineering, cloud computing and green computing.

Qiang He received his first Ph.D. degree from Swinburne University of Technology, Australia, in 2009 and his second Ph.D. degree in computer science and engineering from Huazhong University of Science and Technology, China, in 2010. He is a senior lecturer at Swinburne. His research interests include service computing, software engineering, cloud computing and edge computing. More details about his research can be found at https://sites.google.com/site/heqiang/.

Guangming Cui received his Master degree from Anhui University, China, in 2018. He is a Ph.D. candidate at Swinburne University of Technology. His research interests include software engineering, edge computing and service computing.



Phu Lai received his M.Sc. degree in Information Technology in 2017 and is currently working toward a Ph.D. degree at Swinburne University of Technology, Australia. His research interests include software engineering, cloud computing and edge computing.

Mohamed Abdelrazek is an Associate Professor of Software Engineering and IoT at Deakin University. Before joining Deakin University in 2015, he worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). Before 2010, he was the head of software development department at Microtech. More details about his research can be found at https://sites.google.com/site/mohamedalmorsy/.

John Grundy received the B.Sc. (Hons), M.Sc., and Ph.D. degrees in computer science from the University of Auckland, New Zealand. He is currently a professor of software engineering at Monash University, Melbourne, Australia. He is an associate editor of the IEEE Transactions on Software Engineering, the Automated Software Engineering Journal, and IEEE Software. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at https://sites.google.com/site/johncgrundy/.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. Jin received his Ph.D. in computer engineering from HUST in 1994. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

3.2 Multi-data Placement Strategies in Constrained Edge Computing

In **Chapter 3.1**, we investigate the individual data placement strategies with the aim to place a single data on edge servers to cover all the app users in a specific area at minimum data caching cost. However, three major issues have not been considered properly in **Chapter 3.1**. **First**, an app vendor may want to place multiple data for its users in the same area. For example, YouTube may want to place multiple popular videos requested by a lot of users. **Second**, edge servers' storage capacities are constrained and must be reserved by service providers for placing their data. **Third**, an app user may be able to retrieve data from edge servers via multiple hops over the edge server graph, instead of just zero or one hop. In this chapter, we study this multi-data placement problem from the app vendor's perspective to address the above three issues. We first model this problem and prove its NP-hardness. To solve this problem, we propose two approaches, including an optimal approach to solve this problem exactly and an approximation approach to find the near-optimal solutions to large-scale scenarios. To evaluate the proposed approaches, we conduct extensive experiments based on a real-world dataset.

This chapter is based on a published paper, entitled: Constrained App Data Caching over Edge Server Graphs in Edge Computing Environment, *IEEE Transactions on Services Computing*, 2021. DOI:10.1109/TSC.2021.3062017.

Constrained App Data Caching over Edge Server Graphs in Edge Computing Environment

Xiaoyu Xia, Feifei Chen, John Grundy, *Senior Member, IEEE*, Mohamed Abdelrazek, Hai Jin, *Fellow, IEEE*, and Qiang He*, *Senior Member, IEEE*

Abstract—In recent years, edge computing, as an extension of cloud computing, has emerged as a promising paradigm for powering a variety of applications demanding low latency, e.g., virtual or augmented reality, interactive gaming, real-time navigation, etc. In the edge computing environment, edge servers are deployed at base stations to offer highly-accessible computing capacities to nearby end-users, e.g., CPU, RAM, storage, etc. From a service provider's perspective, caching app data on edge servers can ensure low latency in its users' data retrieval. Given constrained cache spaces on edge servers due to their physical sizes, the optimal data caching strategy must minimize overall user latency. In this paper, we formulate this Constrained Edge Data Caching (CEDC) problem as a constrained optimization problem from the service provider's perspective and prove its $\mathcal{NP}$-hardness. We propose an optimal approach named CEDC-IP to solve this CEDC problem with the Integer Programming technique. We also provide an approximation algorithm named CEDC-A for finding approximate solutions to large-scale CEDC problems efficiently and prove its approximation ratio. CEDC-IP and CEDC-A are evaluated on a real-world data set. The results demonstrate that they significantly outperform four representative approaches.

Index Terms—edge computing, data caching, optimization, approximation algorithm

1 Introduction

The world has witnessed an exponential growth of mobile devices including mobile phones, wearable devices, tablets, smart vehicle and Internet-of-Things (IoT) devices [1]. These devices introduce massive traffic that leads to network congestion and significantly impacts the quality of service, especially service latency, which has become the major obstacle to latency-sensitive applications such as virtual or augmented reality, interactive gaming, real-time navigation [2]. *Edge computing* is proposed to tackle this challenge, where *edge servers* are attached to base stations or access points close to users to offer them computation and storage resources at the edge of the network [3]. It is a key technology that facilitates the 5G mobile network [4]. In the edge computing environment, edge servers, each powered by one or more physical machines, are deployed at base stations or access points that are geographically close to app users. Service providers can hire computing capacities on edge servers and host their applications on edge servers (referred to as *edge apps* hereafter) to ensure low latency for their app users [5]. In the meantime, computation tasks can be offloaded from app users' devices to their nearby edge servers to reduce the computation consumption and save energy on their devices [6], [7].

As an increasing number of mobile and IoT devices begin to access edge apps, more app data will be transmitted through edge servers between the cloud and the app users. From a service provider's perspective, caching those app data, especially the popular ones like viral videos and photos from social web apps such as Facebook, will minimize the delay in app users' data retrieval. They can retrieve app data from nearby edge servers instead of from the cloud if those data are already cached on those edge servers. In addition, caching app data on edge servers can also considerably reduce the amount of data transferred from the cloud to app users, and consequently lower the service providers' cost of data transfer under the pay-as-you-go pricing scheme [8].

Data caching techniques have been widely employed in many domains, e.g., web [9], database [10]. In the network domain, data caching has also been intensively studied to leverage its advantages, i.e. saving bandwidth consumption, reducing network latency and reducing energy consumption. In the last few years, many researchers have investigated caching strategies from different perspectives, e.g., coded caching [11], request routing [12], and information theoretic caching [13]. As a new distributed computing paradigm, edge computing offers new opportunities and raises critical challenges for data caching. The fundamental objective and mechanism are to cache popular app data on edge servers so that nearby app users can retrieve the cached app data with low latency. This is especially important for latency-sensitive applications, e.g., interactive web gaming such as Google Stadia, social video streaming such as TicTok, etc. Caching app data on edge servers can also lift the traffic burden on the Internet backbone by reducing the

- *X. Xia, F. Chen and M. Abdelrazek are with School of Information Technology, Deakin University, Geelong, Victoria, Australia. E-mail: xiaoyu.xia@deakin.edu.au; feifei.chen@deakin.edu.au; mohamed.abdelrazek@deakin.edu.au.*
- *J. Grundy is with Faculty of Information Technology, Monash University, Melbourne, Victoria, Australia. E-mail: john.grundy@monash.edu.*
- *H. Jin is with School of Computer Science and Technolgoy, HuaZhong University of Science and Technology, China. Email: hjin@hust.edu.cn.*
- *Q. He is with School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria, Australia. E-mail: qhe@swin.edu.au.*

Manuscript received March xx, 202x; revised June xx, 202x.

network traffic data significantly [14].

From the service provider's perspective, edge data caching (EDC) aims to cache a single data on edge servers to cover all the app users in a specific area at minimum data caching cost [15], [16]. However, three major issues have not been considered properly by the existing work. **First**, a service provider may want to cache multiple data for its users in the same area. For example, YouTube may want to cache multiple popular videos requested by a lot of users. **Second**, edge servers' constrained storage capacities are constrained and must be reserved by service providers for caching their data. Unlike cloud servers that have access to virtually unlimited storage capacities in the cloud, edge servers' storage capacities are limited due to its size limit [6], [17], [18], [19]. In the open edge computing environment, many service providers may need to hire storage capacities on the edge servers in the same area for caching their data. This causes fierce competition among service providers and makes it practically impossible for every service provider to cache a huge amount of data on every edge server. In such an environment, a common practice is for service providers to reserve storage spaces on edge servers for caching their data. Thus, cost-effectiveness is a key factor in the formulation their data caching strategies. Unlike the edge infrastructure provider who often aims to serve all the users, service providers pursue to maximize caching benefits by fully utilizing the reserved caching spaces. Full user coverage is not always mandatory. **Third**, an app user may be able to retrieve data from edge servers via multiple hops over the edge server graph, instead of just zero or one hop as constrained in [15]. Based on the edge-cloud architecture [20], adjacent edge servers deployed at different base stations can communicate with each other and transmit data via high-speed links [6], [21], [22]. Thus, an app user can access data cached on an edge server via multiple hops over the edge server graph. However, to ensure low data retrieval latency for its users, a service provider must specify its app-specific latency constraint by the maximum number of hops via which its user can retrieval data from an edge server over the edge server graph.

To summarize, in practice, a service provider usually reserves some caching spaces on edge servers - depending on its caching budget - to cache multiple popular data on edge servers in an area for its users to access under the latency constraint. An *optimal data caching strategy* must minimize its app users' data retrieval latency with constrained hired storage spaces on edge servers. In this paper, this problem is referred as the *Constrained Edge Data Caching* (CEDC) problem. We study this problem from the service provider's perspective to address the above three issues. The major contributions of this paper are:

- We model and formulate the CEDC problem as a constrained optimization problem (COP) from the service provider's perspective.
- We prove that the CEDC problem is $\mathcal{NP}$-hard based on the weighted k-set packing problem.
- We develop an optimal approach, namely CEDC-IP, for solving the CEDC problem exactly with the Integer Programming technique.
- We develop an approximation approach named CEDC-A for finding approximate solutions to large-scale CEDC problems efficiently and prove the approximation ratio.
- We conduct extensive experiments on a real-world data set to evaluate the proposed approaches against four representative approaches.

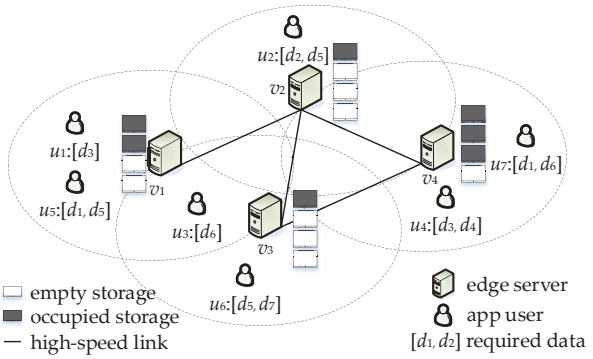


Fig. 1. An example CEDC scenario

The rest of the paper is organized as follows. Section 2 presents an example to illustrate and motivate the CEDC problem. Section 3 formulates the CEDC problem and proves its $\mathcal{NP}$-hardness. Section 4 presents and analyzes our optimal approach and approximation approach for finding solutions to CEDC problems. Section 5 experimentally evaluates the proposed approaches. Section 6 reviews the related work. Section 7 concludes this paper and points out our key future work.

2 Motivating Example

In the edge computing environment, adjacent edge servers deployed at different base stations and access points can communicate with their neighbor edge servers and share their storage resources via high-speed links [6], [21]. Thus, the edge servers in a particular area constitute an *edge server network*. It can be modeled as a graph where a node represents an edge server and an edge represents the link between two edge servers. Data caching in an edge server network differs from data caching in the cloud computing environment as well as other conventional distributed computing environments with its three unique constraints, i.e.,*server capacity constraint*, *server coverage constraint* and *server adjacency constraint*.

Server Capacity Constraints: The storage resources on an edge server are usually limited due to its size limit [3], [15]. The competition between service providers makes it impossible for a service provider to cache all app data on every edge server. Thus, the common practice is for service providers to reserve certain cache spaces on edge servers for caching popular app data.

Server Coverage Overlaps: To avoid any blank coverage areas in a specific geographic area, the coverage areas of nearby edge servers often intersect [23], [24]. Thus, app users in an overlapping area can access any of the edge servers covering them.

Server Adjacency Constraints: An app user can retrieve a piece of app data from its nearby edge servers – we refer to as *local edge servers* hereafter – that cover the app user if the app data is cached on one of these edge servers. If the app data is not cached on any of those local edge servers, the app user can retrieve it from other edge servers that are linked to its local edge servers via multiple hops over the edge server graph – we refer to as *neighbor edge servers* – under the latency constraint. Either way, the service provider's latency constraint will ensure that it is faster than retrieving the app data from a server in the remote cloud.

From the service provider's perspective, the objective of CEDC is to minimize its users' overall data retrieval latency by caching app data with limited reserved cache spaces on edge servers in a specific area. A representative CEDC scenario is caching viral videos and photos for social web apps. Social app users such as Facebook or Instagram users access popular videos and photos shared by either their friends or public figures. Always transmitting data from the cloud to individual app users creates immense pressure on the network and increases the latency in their data retrieval, especially in areas with high user density and dynamic traffic conditions. Caching those data on edge servers brings them much closer to the users and reduce the latency in their data retrieval.

Example 1: Fig. 1 presents an example area with four edge servers, i.e., $\{v_1, ..., v_4\}$, each covering a specific geographic area. The boxes by each edge server represent the cache spaces hired by the service provider on that edge server. To illustrate the CEDC problem generically, each box in Fig. 1 can cache one piece of app data. From the service provider's perspective, caching all its popular data on every edge server in the area can easily accommodate all its users in the area. However, this is not cost-effective nor practical due to the server capacity constraint discussed above. Due to the server coverage constraint and the server adjacency constraint, the data must be cached on a number of those edge servers so that all the users in that area can retrieve the data from either their local edge servers or neighbor edge servers. For example, we assume that the service provider' latency constraint in Fig. 1 is 1 hop. This allows an app user to access any edge servers within 1 hops over the edge server graph for cached app data. Otherwise, it will have to retrieve it from the service provider's remote cloud server. For example, u_1 and u_5 can only retrieve cached data from either their local edge server v_1 or their neighbor edge server v_2. Apparently, there are multiple data caching strategies that fulfill all the three constraints. **The one that minimizes the users' overall data retrieval latency is the optimal solution to the CEDC problem**. Thus, the CEDC problem is inherently a constrained optimization problem (COP).

Model: To quantify the optimization objective and constraints in the CEDC problem in a generic manner, we model the data sizes and cache spaces by the number of data units, and the data retrieval latency by the number of hops. Take Fig. 1 as an example. Each piece of data to be cached is treated as 1 unit, and the total number of cache spaces on v_1 is 4 units, 2 of which are available at the moment. This way, these models can be easily extended for calculating data caching cost given the data size and a specific pricing model, e.g., caching cost per size. Let us still assume that the service provider's latency constraint in this scenario is 1 hop, and user u_7 requests for data d_1 and d_6. The data retrieval latency consists of two major components: the latency between the user and its local edge server(s), and the latency between edge servers. The former component is not avoidable and thus is not considered in the formulation of data caching strategy. Thus, u_7's data retrieval latency is 0 hop if the requested data is cached on edge server v_4, or 1 hop if it is cached on v_2 or v_3. If the data is only cached on edge server v_1, u_7 cannot retrieve the data from any edge server in this area without violating the latency constraint. Similar to the generic models for data and cache spaces, the generic latency model can be extended with different latency specifications.

The model and approaches proposed in this research are generic and applicable to various apps. In our model data are cached on edge servers in whole and we do not consider the situation where data can be partially cached, e.g., video segments. In addition, the scale of the CEDC problem in real-world scenarios can be much larger than the example presented in Fig. 1. Finding an optimal solution to such a CEDC problem is not trivial. Similar to many studies of edge computing [6], [21], [23], [25], [26], [27], [28], [29], we investigate the CEDC problem in quasi-static scenarios where the app users remain unchanged during the data retrieval, e.g., their data needs and locations. More dynamic scenarios will be investigated in our future work.

3 Problem Formulation

3.1 Problem Statement

In this research, we model the networked n edge servers in a specific area as a graph $G(V, E)$ where V is the set of nodes and E is the set of edges in G. In this graph, each node $v_i \in V$ represents an edge server, while each edge $e_t \in E$ represents an edge between two nodes in G. *In the remainder of this paper, we will speak inter-changeably of an edge server and its corresponding node in G, both denoted by v.* The notations adopted in the paper are summarized in Table 1.

As described in Section 2, we formulate the CEDC problem in a generic manner by measuring data retrieval latency by the number of hops between edge servers and data sizes and spaces by the number of data units.

Given a set of data $D = \{d_1, ..., d_k\}$ to be cached on the edge servers in the area, a data caching strategy is a vector $R = \{\langle r_1^1, ..., r_n^1\rangle, ..., \langle r_1^k, ..., r_n^k\rangle\}$, where $r_i^f \in \{0, 1\} (1 \leq i \leq n, 1 \leq f \leq k)$ denotes whether data d_f is cached on edge server v_i.

As discussed in Section 2, the service provider has reserved a finite amount of cache spaces on each edge server. Thus, the number of data cached on an edge server v_i cannot exceed its available cache spaces as_i:

$$\sum_{d_f \in D} r_i^f \leq as_i \tag{1}$$

The distance between two nodes in the graph is measured by their shortest path. As we use the number of hops to measure the data retrieval latency, the latency in an app user u's retrieval of data f is measured as follow:

$$l_u^f = \min\{l_{i,j}, r_j^f = 1, v_j \in V\}, \forall u \in U_i \tag{2}$$

TABLE 1
Summary of Notations

Notation	Description
as_i	available cache spaces on edge server i
b_u^f	maximum benefit for user u retrieving d_f
$b_{u,j}$	benefit of caching replica on server v_j for app user u
D	finite set of data
d_f	data f
E	set of links between edge servers
G	graph presenting a particular area
k	total number of data in D
L	latency constraint from service provider
$l_{i,j}$	latency from server i to server j
l_u^f	minimum latency from app user u to retrieve d_f
m	total number of app users
n	total number of edge servers
R	set of binary variables $\langle r_1^f, ..., r_n^f \rangle$ indicates whether data d_f is cached on edge servers from v_1 to v_n
r_i^f	binary variable indicating cache d_f on edge server v_i
t_u^f	binary variable indicating user u requires data d_f
U	set of app users
U_i	set of users covered by server v_i
V	set of edge servers
v_i	edge server i

where $l_{i,j}$ is the number of hops between v_i and v_j.

To evaluate and compare the effectiveness of different data caching strategies, the concept of data caching benefit is introduced here. It is calculated based on the latency reduction in user data retrieval. We use the number of hops reduced by cached data on an edge server to measure the data caching benefit. Given a latency constraint value L, the data caching benefits is 0 when the latency achieves $L+1$ hops. Denote $l_T = L+1$ as the value breaking the threshold, the following equation shows how to calculate the benefit $b_{u,j}$ produced for app user $u \in U_i$ if edge server v_j is selected to cache the data:

$$b_{u,j} = \max\{l_T - l_{i,j}, 0\} \tag{3}$$

Example 2: Take Fig. 1 as an example. Let us assume $L = 1$ and edge server v_1 is selected to cache data d_5. This way, $L_T = 2$ and we can calculate the benefits for user u_2, u_5 and u_6 to obtain d_5 from v_1, where $b_{2,1} = 1$, $b_{5,1} = 2$ and $b_{6,1} = 0$.

As discussed in Section 2, to avoid the blank area that are not covered by any edge servers, the coverage of nearby edge servers often partially overlap [3]. An app user in the overlapping area can access multiple optional local edge servers and neighbor edge servers for cached data. Thus, the data caching benefit produced by the data caching strategy for an app user u to retrieve data f is:

$$b_u^f = \max\{r_j^f \cdot b_{u,j} \cdot t_u^f, v_j \in V, t_u^f \in \{0,1\}\} \tag{4}$$

where the binary variable t_u^f indicates whether user u requires d_f.

From the service provider's perspective, the optimization objective is to maximize the total reduction in all users' overall data retrieval latency produced by its data caching strategy R, which can be converted to the maximization of the total caching benefit of all users based on (4):

$$\text{maximize } benefit(R) \tag{5}$$

3.2 Problem Hardness

In this section, we demonstrate that the COP of CEDC is $\mathcal{NP}$-hard by proving the following theorem.

Theorem 1. *The COP of CEDC is $\mathcal{NP}$-hard.*

Proof To prove this problem is $\mathcal{NP}$-hard, we first introduce the weighted k-set packing problem (WKSP). The WKSP problem is known to be $\mathcal{NP}$-hard [30]. Given a universe U_e with elements $\forall e \in U_e$, a set S of subsets of U_e and an integer number k. The subset C is a packing, where $C \subseteq S$. All sets $s \subseteq C$ are pairwise disjoint. Let $weight(s)$ be the weight of the set s and k be the maximum number of selected sets. The formulation is displayed below:

$$object : \max \sum_{s \in C} weight(s) \cdot X_s \tag{6a}$$

$$s.t. : \sum_{s \in S} X_s \leq k \tag{6b}$$

$$X_s \in \{0,1\}, \forall s \in S \tag{6c}$$

$$\sum_{e \in U_e} X_e \leq 1 \tag{6d}$$

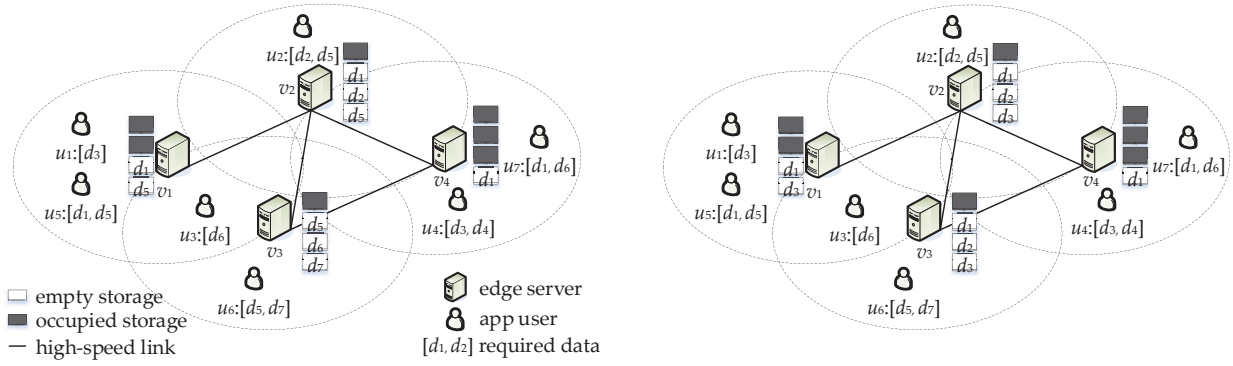
Now we prove that the WKSP problem can be reduced to an instance of the CEDC problem. We define the elements based on the data requests and the users. For example, u_1 require data $\{d_1, d_2\}$, u_2 requires $\{d_2\}$ and u_3 requires $\{d_1\}$. In this case, we can define the elements $e \in \{t_1^1, t_1^2, t_2^2, t_3^1\}$. The reduction can be done as follows: given an instance $WKSP(S, U_e, k, weights(s))$, we can construct the set of $|S|$ servers, denoted by V, and the set of $|U_e|$ users, denoted by U. Let $n = k$, we can construct an instance $CEDC(V, U, n, benefit(v))$ with the reduction above in polynomial time, where function $benefit(v_f)$ is calculated as the sum of benefit if data f is cached on edge server v. As the constraint (6b) restricts the total number of selected sets, we can project the equation $\sum_{v_i \in V} \sum_{d_f \in D} r_i^f \leq \sum_{v_i \in V} as_i$ based on (1) in the CEDC problem to that constraint, such as $benefit(v_i) = 0$ if there is no available cache spaces on edge server i. Based on (3) and (4), data f requested by user $u \in U$ can only be calculated once into the data caching benefit. Thus, constraint (6d) can be fulfilled. Moreover, the increase in the benefit produced by caching data f on edge server v can be projected to $weight(s)$. In this case, any solution R satisfying objective (6a) also satisfies objective (4).

In conclusion, any solution $\mathcal{Y}$ always satisfies the reduced CEDC problem if $\mathcal{Y}$ satisfies the WKSP problem. Therefore, the CEDC problem is reducible from the WKSP problem and it is $\mathcal{NP}$-hard.

□

4 Edge Web Data Caching Strategy Formulation

We first model the CEDC problem with the Integer Programming technique, then prove that the CEDC problem is $\mathcal{NP}$-hard. After that, we propose an approximation algorithm for finding approximate solutions to large-scale CEDC problems efficiently and prove its approximation ratio.



(a) Strategy R_1 (b) Strategy R_2

Fig. 2. Example CEDC Strategies

4.1 Optimization Approach

From the service provider's perspective, the solution to the CEDC problem must maximize the data caching benefit measured by the total reduction in all the app users' data retrieval latency in this area. In the meantime, the server capacity constraint, server coverage constraint and server adjacency constraint must be fulfilled. Thus, the CEDC problem can be modeled as a constrained optimization problem (COP). The COP model for the CEDC problem is formally expressed as follows.

For a graph $G = (V, E)$, where $V = \{v_1, .., v_n\}$, there is a matrix of variables $R = \{\langle r_1^1, ..., r_n^1\rangle, ..., \langle r_1^k, ..., r_n^k\rangle\}$, where $r_i^f \in \{0, 1\}, \forall i \in \{1, ..., n\}, \forall f \in \{1, ..., k\}$, r_i^f being 1 if a data replica of d_f is cached on the i^{th} node, 0 otherwise. The constraints for the COP model are:

$$b_u^f = \max(r_i^f \cdot b_{u,i} \cdot t_u^f) \quad \forall i \in \{1, ..., n\}, \forall u \in \{1, ..., m\}, \forall f \in \{1, ..., k\} \tag{7}$$

$$\sum_{f=1}^{k} r_i^f \leq as_i, \forall i \in \{1, ..., n\}, \forall f \in \{1, ..., k\} \tag{8}$$

Constraint family (7) is converted from (4). It ensures that every app user will always retrieve the required data from the nearest possible edge server. Constraint family (8) guarantees that the data cached on each edge servers must not exceed the available cache spaces.

As discussed in Section 3.1, there might be multiple solutions to this COP fulfilling (7) and (8).

Example 3: In Fig. 1, there are multiple possible solutions satisfying constraints (7) and (8). Assuming the latency constraint is no more than 1 hop in this scenario, two possible data caching strategies are presented in Fig. 2: $R_1 = \{\langle 1, 0, 0, 0, 1, 0, 0\rangle, \langle 1, 1, 0, 0, 1, 0, 0\rangle, \langle 0, 0, 0, 0, 1, 1, 1\rangle, \langle 1, 0, 0, 0, 0, 0, 0\rangle\}$ that caches data $\{d_1, d_5\}$ on v_1, $\{d_1, d_2, d_5\}$ on v_2, $\{d_5, d_6, d_7\}$ on v_3 and $\{d_1\}$ on v_4, and $R_2 = \{\langle 1, 0, 1, 0, 0, 0, 0\rangle, \langle 1, 1, 1, 0, 0, 0, 0\rangle, \langle 1, 1, 1, 0, 0, 0, 0\rangle, \langle 1, 0, 0, 0, 0, 0, 0\rangle\}$, caching data $\{d_1, d_3\}$ on v_1, $\{d_1, d_2, d_3\}$ on v_2, $\{d_1, d_2, d_3\}$ on v_3 and $\{d_1\}$ on v_4. Both of R_1 and R_2 fulfill constrains (7) and (8). However, the overall caching benefits produced by R_1 and R_2 are different based on (4) and (7): $benefit(R_1) = 17$ and $benefit(R_2) = 9$. Thus, the below objective function that maximizes the benefit of caching data D over G is included in the COP model to achieve the service provider's optimization objective:

$$\max \sum_{u=1}^{m} \sum_{f=1}^{k} b_u^f \tag{9}$$

The COP above can be solved with Integer Programming problem solvers, such as Gurobi[1] and IBM CPLEX Optimizer[2]. This optimal approach is named CEDC-IP hereafter. Specific caching cost models and network latency models can be easily integrated to this CEDC-IP model.

4.2 Approximation Algorithm

As the COP of CEDC is $\mathcal{NP}$-hard, finding the optimal solution to the COP is intractable in large-scale CEDC scenarios. This section presents an approximation algorithm, named CEDC-A, for finding approximate solutions to large-scale CEDC problem efficiently.

Given $V = \{v_1, ..., v_n\}$, $U = \{u_1, ..., u_m\}$ and $D = \{d_1, ..., d_f\}$, CEDC-A creates an initial candidate list , and then implements an iterative process for each initial candidate. After the above processes, CEDC-A selects the candidate solution with the maximum benefit for a service provider to cache data on selected edge servers. The pseudo code is presented in Algorithm 1, while the functions used in Algorithm 1 are presented in Algorithm 2 and 3.

In this algorithm, the decision to cache one kind of data on an edge server is treated as a $candidate$. The algorithm starts with the initialization in Lines 1-3. The algorithm initiates the set of solution candidates, C, by calculating the benefit increment of each data in each edge server's coverage area (Line 3). Each candidate has two properties, i.e., server id and data. CEDC-A always selects the candidate with maximum benefit value to cache candidate's data on candidate's server based on $c_i \in C$ (Lines 4 to 12). In the end, the solution with the highest benefits will be selected as the result of CEDC-A.

The computational complexity of functions presented in Algorithm 2 and 3 is $O(kn)$. Moreover, $initCandidates()$ can produce at most kn candidates, and the used cache

1. http://www.gurobi.com/
2. https://www.ibm.com/analytics/cplex-optimizer

Algorithm 1 CEDC-A Algorithm

Input: availableSpaces, U, V, D
$C \leftarrow \varnothing$
C = initCandidates()
for each $c_i \in C$ **do**
  as = copy(availableSpaces)
  candidate = getMaxBenefitCandidate(c_i)
  while |as| != 0 & candidate != null **do**
    $c_i \leftarrow c_i \bigcup$ candidate
    $as_{candidate.id} = as_{candidate.id} - 1$
    candidate = getMaxBenefitCandidate(c_i)
  end while
end for
$R = \arg\max_{c_i \in C}$ benefit(c_i);
return R

Algorithm 2 Function initCandidates

initCandidates():
Candidates $\leftarrow \varnothing$
for each $v_i \in V$ **do**
  if $as_i! = 0$ **then**
    for each $d_f \in D$ **do**
      Candidates $\leftarrow$ Candidates $\bigcup \{r_i^f\}$
    end for
  end if
end for
return Candidates

Algorithm 3 Function getMaxBenefitCandidate

getMaxBenefitCandidate(c_i):
candidate = null
for each $v_j \in V \bigcap \neg c_i$ **do**
  if $as_j! = 0$ **then**
    for each $d_f \in D$ **do**
      if benefit($c_i \bigcup$ candidate) $\leq$ benefit($c_i \bigcup r_j^f$) **then**
        candidate = new $Candidate()$
        candidate.id = j
        candidate.data = f
      end if
    end for
  end if
end for
return candidate

spaces on all edge servers are also at most kn. Thus, in the worst-case scenario, the computational complexity of Algorithm 1 is $O(k^2n^2)$.

Now, we prove the approximation ratio of CEDC-A, where it is the ratio of benefits produced by CEDC-A and that produced by optimal solution in the worst cases.

As function $initCandidates()$ provides the candidate list with the first cache decision, we can treat this function as the first iteration in Algorithm 1. Let OPT present the optimal solution (found by CEDC-IP) of the CEDC problem and $benefit(OPT)$ denote the benefit obtained by the OPT caching strategy. Let us assume that the order of cache decisions made by OPT is ascending in terms of cache benefit, and γ is the index of the iteration when the first edge server included in OPT's strategy but not in CEDC-A's strategy R.

Theorem 2. For each iteration $t \leq \gamma$, the following inequality is satisfied:

$$benefit(OPT) - benefit(R_{t-1}) \leq |as| \cdot \Delta b_t \tag{10}$$

where R_{t-1} is the strategy in iteration $t-1$ and Δb_t is the benefit produced by including r_t into strategy R.

Proof Since R_t selects the edge server with the maximum benefit at the t^{th} iteration, for each edge server in OPT but not in R_{t-1}, the benefit increment is at most Δb_t. Since there are a maximum of $|as|$ available cache spaces, the total benefit produced by the edge servers in OPT but not R_{t-1} is at most $|as| \cdot \Delta b_t$. Thus, the above inequality is satisfied. □

Theorem 3. For each iteration $t > \gamma$, the benefit produced by R_t fulfills:

$$benefit(R_t) \geq \left(1 - \left(1 - \frac{1}{|as|}\right)^{t-1}\right) benefit(OPT) \tag{11}$$

Proof Based on Theorem 2, the benefit achieved by S_t can be calculated by (12). Thus, we can easily prove (11) by the inductive proof. The details of the proof process are omitted here.

$$\begin{aligned} &benefit(R_t) = benefit(R_{t-1}) + \Delta b_t \\ &\geq benefit(R_{t-1}) + \frac{benefit(OPT) - benefit(R_{t-1})}{|as|} \\ &= \left(1 - \frac{1}{|as|}\right) benefit(R_{t-1}) + \frac{1}{|as|} benefit(OPT) \end{aligned} \tag{12}$$

□

Theorem 4. The approximation ratio of CEDC-A is $\frac{2}{3}\left(1 - \frac{1}{e}\right)$.

Proof Based on the cache space constraint(8), the algorithm can have at most $|as|$ iterations. Thus, when $t = |as| + 1$, we can obtain (13) based on Theorem 3:

$$benefit(R_{|as|+1}) \geq \left(1 - \frac{1}{e}\right) benefit(OPT) \tag{13}$$

However, this exceeds the constraint (8). As discussed above, the first decision in function $initCandidates()$ is treated as the first iteration.

When $|as| = 1$, the solution obtained by CEDC-A is the same as OPT. In this case, the approximation ratio is 1. When $|as| \geq 2$, the benefit increment by $\Delta b_{|as|+1}$ is less than or equal to $\frac{1}{2} benefit(R_{|as|})$. Thus, there is

$$\begin{aligned} &benefit(R_{|as|}) = benefit(R_{|as|+1}) - \Delta b_{|as|+1} \\ &\geq \left(1 - \frac{1}{e}\right) benefit(OPT) - \frac{1}{2} benefit(R_{|as|}) \end{aligned} \tag{14}$$

Based on (14), the approximation ratio of CEDC-A is $\frac{2}{3}\left(1 - \frac{1}{e}\right)$. □

5 Experimental Evaluation

We experimentally evaluate the performance of CEDC-IP and CEDC-A. All of the experiments were conducted on a Windows-10 machine equipped with Intel Core i7-8550 processor (8 CPUs, 1.80GHz) and 8GB RAM. The COP discussed in Section 4 is solved with IBM's CPLEX Optimizer.

5.1 Baseline Approaches

In the experiments, we evaluate the performance of CEDC-IP and CEDC-A against four representative approaches:

- *Request-based Collaborative Caching (RCC)*: This approach aims to minimize the overall caching cost incurred by serving the most users. This algorithm originated from the collaborative data caching approach in [31], which is implemented in the content delivery network.
- *NCCEDC-IP*: This approach finds the non-collaborative data caching optimal solution, which does not allow collaboration among edge servers. Thus, app users can only access data from their local edge servers. Other than that, it is formulated and implemented in a way similar to CEDC-IP.
- *Greedy-Connection (GC)*: This approach always selects the edge server that has the most neighbour edge servers to cache data under Constraint family (8).
- *Random*: This approach always selects the edge server randomly to cache data under Constraint family (8).

5.2 Experiment Settings

5.2.1 Experiment data

The experiments are conducted on a real-world data set, named EUA data set[3] [3], which is widely used in research on edge computing [23], [32], [33]. This data set contains the geographical locations of 125 base stations and 816 mobile users in the Melbourne CBD area. The links between edge servers are randomly generated to ensure the edge servers constitute a connected graph. In the experiments, L is set to 1 hop. The available cache spaces on each edge server are generated following a normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, where μ is half of the number of maximum cache spaces and σ is 1.

5.2.2 Experimenting parameters

To simulate different CEDC scenarios, three parameters are varied in the experiments.

- Number of edge servers $|\mathbf{V}|$. This parameter impacts the size of graph G and varies from 4 to 14 in steps of 2.
- Number of maximum cache spaces **MS**. This parameter impacts the available cache spaces on edge edge server and varies from 2 to 10 in steps of 2.
- Number of data $|\mathbf{D}|$. The total number of data to be cached over G. This parameter varies from 3 to 8 in steps of 1.

3. https://github.com/swinedge/eua-dataset

TABLE 2
Parameter Settings

	$\|\mathbf{V}\|$	**MS**	$\|\mathbf{D}\|$
Set #1.1	4, 6, 8, 10, 12, 14	4	6
Set #1.2	10	2, 4, 6, 8, 10	6
Set #1.3	10	4	3, 4, 5, 6, 7, 8
Set #2.1	20, 30, 40, 50, 60, 70	10	15
Set #2.2	50	2, 4, 6, 8, 10	15
Set #2.3	50	10	5, 10, 15, 20, 25, 30

5.2.3 Performance Metrics

In this experiments, three metrics are employed to evaluate the performance of the approaches, one for effectiveness and one for efficiency:

- **Benefit per cache cost (**bpc**)**, measured by the total number of hops reduced divided by the data cache cost, higher the better.
- **Served request ratio per cache cost (**$SRRpc$**)**, measured by the ratio of the served data requests divided by the reserved data cache cost, higher the better.
- **Computational overhead (**$time$**)**, measured by the time taken to find the solution, the lower the better.

Table 2 summarizes the parameter settings. There are two main sets of experiments, Set #1 for small-scale experiments and Set #2 for large-scale experiments. All six approaches are implemented in Set #1, while CEDC-IP and NCCEDC-IP are not implemented in Set #2, because they cannot find an optimal solution to the $\mathcal{NP}$-hard CEDC problem in Set #2 within a reasonable time. Every time the value of a parameter varies, the experiment is repeated for 100 times and the averaged results are reported. To isolate the impact of the number of app users, we randomly select 40 covered app users and 200 covered app users from the data set in each run of the experiments in Set #1 and Set #2, respectively.

5.3 Experimental Results

Table 3 summarizes the results of experiment Set #1 and Set #2 where the best and second-best performances are marked as dark and light grey, respectively, in each column.

5.3.1 Impact of number of edge servers

The results of experiment Set #1.1 are presented in Fig. 3(a) and Fig. 4(a). It shows that the **benefit per cache cost and the served request ratio per cache cost achieved by CEDC-IP and CEDC-A outperform the other approaches significantly**. In Fig. 3(a), when the number of edge servers increases from 4 to 14, the benefit per cache cost achieved by all six approaches decreases, from 13.16 to 4.34 by 67.02% for CEDC-IP, from 12.57 to 4.20 by 66.59% for CEDC-A, from 10.99 to 3.71 by 66.24% for RCC, from 10.35 to 3.59 by 65.31% for NCCEDC-IP, from 9.54 to 3.34 by 64.99% for GC and from 7.89 to 3.06 by 61.21% for Random. The advantages of CEDC-IP are 3.77% over CEDC-A, 18.47% over RCC, 25.53% over NC, 38.12% over GC and 62.76% over Random on average. It is also shown in Fig. 3(a) that **the advantages of CEDC-IP and CEDC-A increase with the increase in**

TABLE 3
Average performance results

Approaches	Set #1.1			Set #1.2			Set #1.3		
	bpc	*SRRpc* (%)	*Time*	*bpc*	*SRRpc* (%)	*Time*	*bpc*	*SRRpc* (%)	*Time*
CEDC-IP	7.4817	7.5450	1.6518	4.7040	4.6500	0.6380	5.9150	5.7500	10.6384
CEDC-A	7.2100	7.6933	0.0002	4.5840	4.7000	0.0003	5.7183	5.7233	0.0002
RCC	6.3150	6.0817	0.0001	4.1880	3.9660	0.0002	5.2250	5.2250	0.0001
NCCEDC-IP	5.9600	6.7217	0.4198	3.8820	4.1940	0.3318	4.9483	5.2517	1.1959
GC	5.4167	5.0900	0.0002	3.6700	3.4820	0.0002	4.5917	4.3300	0.0001
Random	4.5967	4.4217	0.0001	3.3580	3.1940	0.0002	4.2467	4.0433	0.0001
Approaches	Set #2.1			Set #2.2			Set #2.3		
	bpc	*SRRpc* (%)	*Time*	*bpc*	*SRRpc* (%)	*Time*	*bpc*	*SRRpc* (%)	*Time*
CEDC-A	3.1288	0.5698	0.0571	3.7530	0.7534	0.0346	2.9696	0.5194	0.0833
RCC	2.7973	0.5025	0.0104	3.1547	0.5889	0.0055	2.6645	0.4590	0.0146
GC	2.5341	0.4527	0.0053	2.4462	0.4632	0.0024	2.6064	0.4550	0.0051
Random	2.3308	0.4202	0.0031	2.1924	0.4166	0.0021	2.4447	0.4298	0.0047

the number of edge servers. Fig. 4(a) shows that the served request ratio per cache cost of CEDC-A is almost the same as that of CEDC-IP. Both CEDC-IP and CEDC-A achieve much higher average served request ratio per cache than the other approaches, i.e., 0.0755 (CEDC-IP) and 0.0769 (CEDC-A) versus 0.0608 (RCC), 0.0672 (NCCEDC-IP), 0.0509 (GC) and 0.0442 (Random).

The experimental results of Set #2.1 are depicted in Fig. 5(a) and Fig. 6(a). Overall, **CEDC-A achieves the highest benefit and served request ratio per cache cost on average**. As shown in Fig. 5(a), CEDC-A outperforms RCC, GC and Random in benefit per cache cost, by 11.85%, 23.47% and 34.24%, respectively. In terms of served request ratio per cache cost, Fig. 6(a) demonstrates that the advantages of CEDC-A are 13.39% over RCC, 25.87% over GC and 35.60% over Random.

With the increase in the number of edge servers (4 to 14 in Set #1.1 and 20 to 70 in Set #2.1), the benefit and served request ratio per cache cost achieved by all approaches decrease. The reason is that, when the number of users is fixed, the maximum benefits and the total number of requests are fixed. Accordingly, the benefit and served request ratio per cache cost decrease when more cache spaces are hired by the service provider with the fixed number of users.

5.3.2 Impact of maximum cache spaces

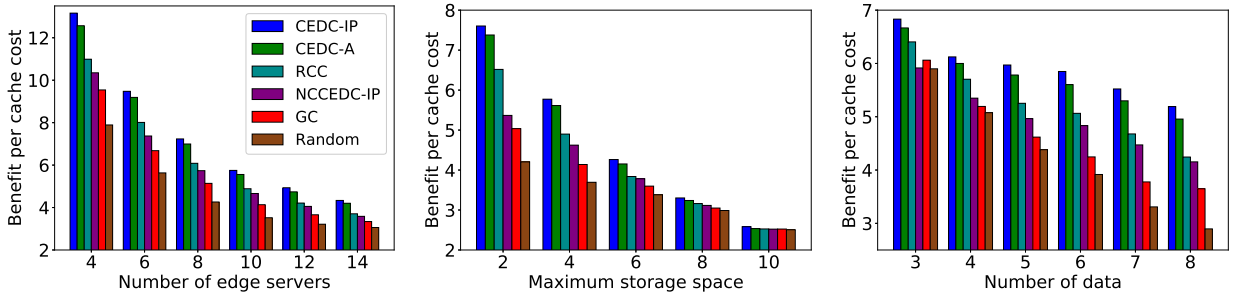
The impacts of the maximum cache spaces on the approaches are shown in Fig. 3(b), Fig. 4(b), Fig. 5(b) and Fig. 6(b). In experiment Set #1.2, **CEDC-IP achieves the highest benefit per cache cost again**, followed by CEDC-A. In terms of the served request ratio per cache cost, CEDC-IP and CEDC-A also outperform the other approaches significantly. In Fig. 3(b), both CEDC-IP and CEDC-A outperform RCC, NCCEDC-IP, GC and Random, by an average of 12.32% and 9.46%, 21.17% and 18.08%, 28.17% and 24.90%, 40.08% and 36.51%, respectively. Moreover, Fig. 4(b) shows that, for the served request ratio per cache cost, the average advantages of CEDC-A are 1.07% over CEDC-IP, 18.61% over RCC, 12.16% over NCCEDC-IP, 35.09% over GC and 47.28% over Random. With the increase in the maximum cache spaces from 2 to 10 in Set #1.2, the cache cost increases. As the number of users is fixed at 40, the maximum total benefits are fixed as well. Thus, the benefit and served request ratio per cache cost decrease for all approaches with the increase in the maximum cache spaces, while the trends and their trends are similar in Fig. 5(b) and Fig. 6(b). In Set #2.2, **CEDC-A achieves the highest benefit and served request ratio per cache cost again**. The advantages of CEDC-A are 18.97% and 27.93% over RCC, 53.42% and 62.65% over GC, 71.18% and 80.84% over Random, in terms of benefit per cache cost and served request ratio per cache cost, respectively.

Based on the results shown in the above figures, **the advantages of CEDC-IP and CEDC-A are more even more significant with fewer available cache spaces**. This indicates that CEDC-IP and CEDC-A are particularly suitable for the edge computing environment. This is because it is a highly competitive environment where the resources on edge servers available for data caching are constrained.

5.3.3 Impact of number of data

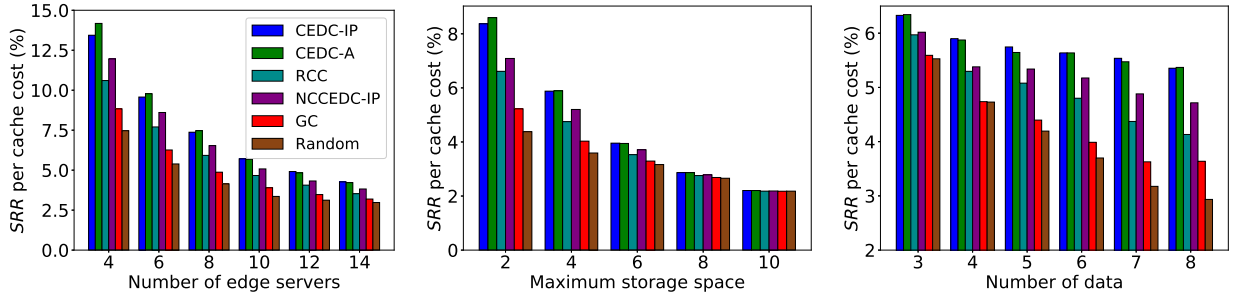
Fig. 3(c) and Fig. 4(c) depict the results obtained in Set #1.3 where the number of data to be cached varies. In terms of the benefit per cache cost and the served request ratio per cache cost, **CEDC-IP and CEDC-A outperform the other approaches with significant margins**. In terms of the benefit per cache cost, as demonstrated in Fig. 3(c), the average advantages of CEDC-IP are 3.44% over CEDC-A, 13.21% over RCC, 19.54% over NCCEDC-IP, 28.82% over GC and 39.29% over Random. In Fig. 5(c) and Fig. 6(c), **the advantage of CEDC-A is significant over the other three approaches**. On average, CEDC-A outperforms RCC by 11.45% and 13.16%, GC by 13.93% and 14.15%, Random by 21.36% and 20.85%, in the benefit per cache cost and served request ratio per cache cost, respectively.

Those results also demonstrate that **the performance of both CEDC-IP and CEDC-A decreases much slower than the other approaches** when the number of data to be cached increases. That is, they scale better with the number of data to be cached.



(a) Set #1.1 (b) Set #1.2 (c) Set #1.3

Fig. 3. Benefit per cache cost vs. parameters in Set #1



(a) Set #1.1 (b) Set #1.2 (c) Set #1.3

Fig. 4. Ratio of served request per cache cost vs. parameters in Set #1

5.3.4 Efficiency

The efficiency of an approach is evaluated by its computational overhead measured by the average time (in seconds) taken to find a solution to the CEDC problem. The results are summarized in Table 3. As shown, **CEDC-IP is much more computationally expensive than all the other approaches** in Set #1. It takes 10.6384 seconds on average in Set # 1.3. This validates the $\mathcal{NP}$-hardness of the CEDC problem - excessive computational overheads are inevitable for finding the optimal solution to large-scale CEDC problems. The other approaches, including CEDC-A, RCC, GC and random, can find a solution almost immediately in Set #1. In Set #2, the execution time increases for all approaches. In Set #2, Random takes the least time to between 0.0021 and 0.0047 seconds on average, while CEDC-A takes the most time, between 0.0346 seconds and 0.0833 seconds. CEDC-A takes more time than RCC, GC and Random. This is the performance price to pay for CEDC-A's effectiveness advantage over these approaches as shown and discussed above. With the significant advantages of CEDC-A, especially where there are more edge servers, more data, and less storage space, it is worth applying CEDC-A in the real deployment.

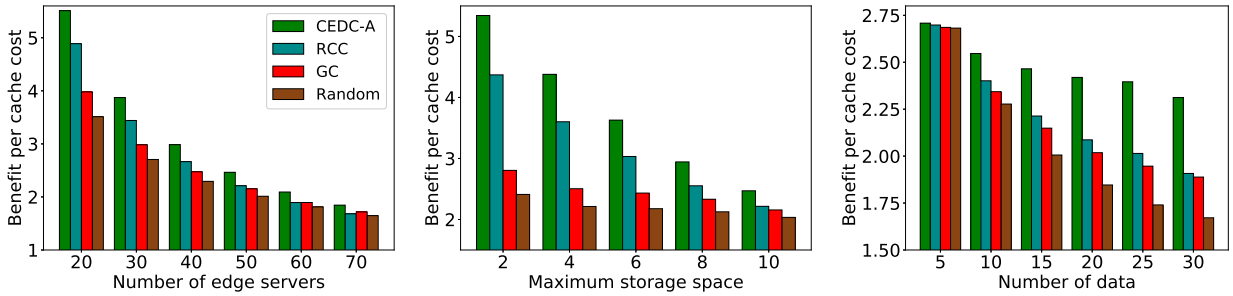
5.3.5 Conclusion

Overall, **CEDC-IP and CEDC-A outperform GC and Random significantly and consistently** in formulating cost-effective data caching strategies in different CEDC scenarios. As an approximation algorithm, the effectiveness of CEDC-A is 96.37% to 97.54% on average as high as CEDC-IP as shown by the results of experiment Set #1. In Set #2, CEDC-A outperforms all other approaches significantly at the price of slightly higher computational overheads. Thus, CEDC-IP is suitable for solving small-size CEDC problems. To solve large-scale CEDC problems, CEDC-A is more practical for its high effectiveness and efficiency in finding near-optimal solutions.

5.4 Threats to Validity

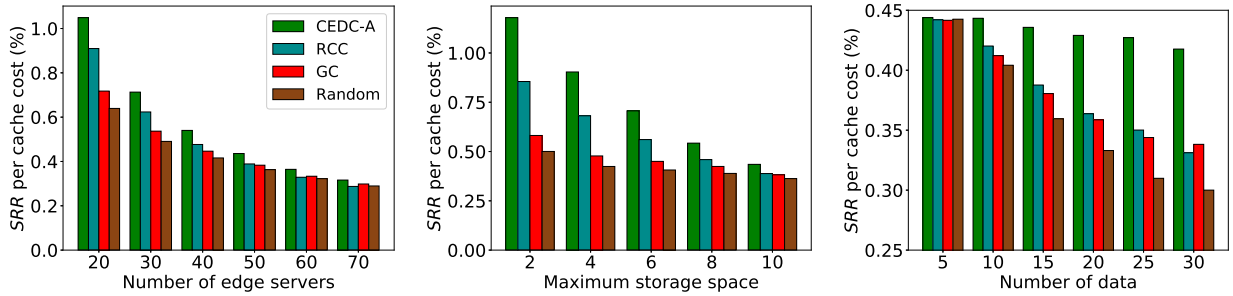
5.4.1 Construct Validity

The main threats to construct validity are the randomly generated graphs and and the four approaches used for comparison in the experiments. The graphs randomly generated in the experiments may not represent all the edge server networks in the real-world edge computing environment. To minimize this threat, the experiment is repeated for 100 times - a total of 100 graphs are randomly generated - every time the value of a setting parameter varies in the experiments. In this way, a large number of edge server networks are simulated in the experiments to provide comprehensive guidelines on the performance of our approaches in real-world scenarios. The comparison to RCC, NCCEDC-IP, GC and Random, may not suffice to comprehensively evaluate CEDC-IP and CEDC-A. To minimize this threat, three experimental parameters are varied in the experiments to simulate different CEDC scenarios. In this way, we could evaluate CEDC-IP and CEDC-A by not only the comparison with the four approaches but also by the impacts of the three varying setting parameters.



(a) Set #2.1 (b) Set #2.2 (c) Set #2.3

Fig. 5. Benefit per cache cost vs. parameters in Set #1



(a) Set #2.1 (b) Set #2.2 (c) Set #2.3

Fig. 6. Ratio of served request per cache cost vs. parameters in Set #1

5.4.2 *Threats to Internal Validity*

The main threat to the internal validity is whether the experiment setting favors CEDC-IP and CEDC-A over other approaches. To minimize this threat, we varied three parameters to simulate various CEDC scenarios so that the performance of different approaches could be compared comprehensively and fairly. Moreover, all experiments were repeated for 100 times and the results were averaged. This way, the results of experiments were somehow set up in a biased manner could be neutralized.

5.4.3 *External Validity*

The main threat here is whether CEDC-IP and CEDC-A are also applicable for other edge computing scenarios. To address this, we formulate the approaches and measure the performance in a more generic way: evaluating the effectiveness by using the number of data replicas and the number of hops for cost and benefit. This way, the exact latency model and cost model can be easily integrated into our approaches. Moreover, the widely-used real-world data set is used to evaluate all approaches. Thus, the representativeness and comprehensiveness of the evaluation are ensured, and this threat is reduced.

6 Related Work

Data caching have been extensively investigated in the fields of conventional distributed computing and cloud computing environments. With the popularity of edge computing, data caching in the edge computing environment is obtaining attention from researchers recently.

6.1 Conventional Distributed Data Caching

In the last few decades, there are many data caching problems investigated in conventional distributed computing environments, including web caching [9], content delivery network [34], etc. Banerjee et al. [35] developed a content placement strategy for information-centric network based on data popularity, namely Greedy Caching. With popular contents cached in the network, the Greedy Caching approach considered the cache miss rate at the edge to decide what contents would be cached on the core server. In [36], the authors formulated two caching strategies for data publish-subscribe systems, including eviction-based caching and time-to-live-based caching to address the space and time issues, respectively. The authors of [37] focused on balancing the trade-off between latency and cost in the content-centric network. They addressed this issue with a holistic model for provisioning the storage capability based on the network performance and the provisioning cost.

6.2 Cloud Data Caching

In the cloud computing environment, a critical problem of data caching is how to utilize cache space efficiently on cloud hosts and mobile devices.

Arteaga et al. [38] proposed CloudCache, a method for managing cache, to fulfill the caching requirement of the workload and minimize cache wear-out. In [39], the authors presented how to use segment access-aware dynamic semantic cache in the cloud computing environment for relational databases. A cache access algorithm was introduced to consider cache exact hit, cache extended

hit, cache partial hit and cache miss. The authors of [40] explored the cache design space for embedded processors with evolutionary techniques for mobile and thin client processors in the cloud computing environment. A heuristic and evolutionary method was presented to generate a near-optimal cache space design for enhancing service quality. In [41], the authors formulated a benefit maximization problem and created a cache replacement approach based on traffic requirements. They also introduced a content clustering method for collecting popular data and clustering similar contents.

6.3 Edge Data Caching

From the perspectives of network topology and infrastructure deployment, edge computing is an extension of cloud computing with distributed computing capacities and services at the edge of the network. App users in various domains can benefit from the advantages of edge computing, e.g., interactive gaming, real-time navigation, augmented reality [2]. Offering many unique advantages, edge computing also raises various new research challenges from the service provider's perspective, e.g., edge user allocation [3], [23], edge data distribution [8], edge application deployment [33], [42], edge data integrity [43], etc.

Recently, there are some researchers starting to investigate the data caching problems in edge computing. As mentioned in Section 2, the data caching strategies from conventional distributed computing and cloud computing cannot be directly applied in edge computing. Thus, those researches introduced new ideas and approaches. Cao et al. proposed an optimal auction mechanism with the consideration of the costs produced during delivery and retrieval [44]. The authors of [45] provided a caching system, namely Agar, from the erasure-coded perspective. Agar was a dynamic programming algorithm which could cache data chunks optimally with consideration of data popularity and network latency. Zhang et al. [46] integrated in-network caching and edge caching to ensure the latency requirements of time-sensitive transmissions over the 5G network. Drolia et al. [47] proposed a caching system, namely Cachier, to minimize the data retrieval latency. They implemented a coordinating mechanism to balance the loads between the cloud server and edge servers dynamically. The authors of [48] introduced a new edge caching architecture with improved resource utility by using smart vehicles as external edge caches. Poularakis et al. [17] studied the joint optimization of service placement and request routing in the edge computing environment. The authors of [49] proposed a hierarchical caching mechanism in the edge computing environment with consideration of wireless communication. They aimed to maximize the hitting rate by caching data in different layers including routers, base stations and mobile devices. In [50], the authors studied the budgeted service placement problem in the edge computing environment. They proposed a Lyapunov-based algorithm for minimizing the overall data retrieval latency. Deng et al. [51] also investigated the service deployment problem in the edge computing environment but tried to minimize the overall cost rather than latency. They provided a primal-dual algorithm named IDA4ReE to solve this problem under a resource constraint and performance requirement. However, these studies did not fully consider the unique constraints in the edge computing environment, i.e. the server capacity constraint, the server coverage constraint and the server adjacency constraint.

Edge computing inherits the pay-as-you-go pricing model from cloud computing, which allows service providers to hire storage resources on edge servers from edge infrastructure providers to cache app data for their own users. Thus, both the benefit produced and the cost incurred by data caching for service providers is critical to the success of edge computing because, after all, service providers are the main customers in the edge computing environment. However, the above studies tackle the data caching problem from either the mobile network operator's or the app user's perspective. In [15], the data caching problem is firstly tackled from the service provider's perspective in the edge computing environment with the aim to cover all the app users in an area. The approach proposed in [15] can only cache data individually and does not consider the constrained cache spaces on edge servers or edge servers' ability to communicate. In this paper, we considered the these practical issues and converted the EDC problem into the constrained edge data caching (CEDC) problem. We solved the CEDC problem in a generic manner to maximize the data caching benefit with finite cache spaces on edge servers, considering the unique server capacity constraint, server coverage constraint and the server adjacency constraint in the edge computing environment.

7 Conclusion

In this paper, we formulated the new constrained edge data caching (CEDC) problem in the edge computing environment from the service provider's perspective. We proved that the CEDC problem is $\mathcal{NP}$-hard. To solve this problem, we proposed an optimal approach named CEDC-IP based on integer programming to maximum the data caching benefit measured by the overall reduction in app users' data retrieval latency with limited cache resources. As the CEDC problem is $\mathcal{NP}$-hard, we also provided an approximation approach named CEDC-A for finding approximate solutions to large-scale CEDC problems efficiently. Extensive experiments were conducted on a widely-used real-world data set to evaluate the performance of the proposed approaches. The results showed that our approaches significantly outperformed the state-of-the-art approaches in various CEDC scenarios. In our future work, we will consider the mobility of app users, real-time cache updating scenarios, security constraints and data regulation.

Acknowledgement

This research is partially funded by Australian Research Council Discovery Projects No. DP180100212, DP200102491 and Laureate Fellowship FL190100035. Qiang He is the corresponding author of this paper.

REFERENCES

[1] A. Osseiran, V. Braun, T. Hidekazu, P. Marsch, H. Schotten, H. Tullberg, M. A. Uusitalo, and M. Schellman, "The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions," in *IEEE 77th Vehicular Technology Conference (VTC2013-Spring)*, 2013, pp. 1–5.

[2] M. Yannuzzi, F. van Lingen, A. Jain, O. L. Parellada, M. M. Flores, D. Carrera, J. L. Pérez, D. Montero, P. Chacin, A. Corsaro *et al.*, "A new era for cities with fog computing," *IEEE Internet Computing*, vol. 21, no. 2, pp. 54–67, 2017.

[3] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *International Conference on Service-Oriented Computing*, 2018, pp. 230–245.

[4] G. Cui, Q. He, X. Xia, P. Lai, F. Chen, T. Gu, and Y. Yang, "Interference-aware saas user allocation game for edge computing," *IEEE Transactions on Cloud Computing*, 2020.

[5] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, vol. 12, no. 4, pp. 29–33, 2017.

[6] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[7] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 123–132.

[8] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 31–44, 2020.

[9] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.

[10] K. Elhardt and R. Bayer, "A database cache for high performance and fast restart in database systems," *ACM Transactions on Database Systems (TODS)*, vol. 9, no. 4, pp. 503–525, 1984.

[11] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. N. Diggavi, "Hierarchical coded caching," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3212–3229, 2016.

[12] M. Dehghan, B. Jiang, A. Seetharam, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 3, pp. 1635–1648, 2017.

[13] C.-Y. Wang, S. H. Lim, and M. Gastpar, "Information-theoretic caching: Sequential coding for computing," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6393–6406, 2016.

[14] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[15] X. Xia, F. Chen, Q. He, G. Cui, P. Lai, M. Abdelrazek, J. Grundy, and H. Jin, "Graph-based optimal data caching in edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 477–493.

[16] ——, "Graph-based data caching optimization for edge computing," *Future generation computer systems*, vol. 113, pp. 228–239, 2020.

[17] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[18] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundancy scheduling for microservice-based applications at the edge," *IEEE Transactions on Services Computing*, 2020.

[19] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, and A. Y. Zomaya, "Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6103–6113, 2020.

[20] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[21] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4514–4526, 2018.

[22] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2020.

[23] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[24] P. Lai, Q. He, G. Cui, X. Xia, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Edge user allocation with dynamic quality of service," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 86–101.

[25] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[26] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 726–738, 2018.

[27] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.

[28] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2018.

[29] J. L. D. Neto, S.-y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: a user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 266–2674, 2018.

[30] E. Hazan, S. Safra, and O. Schwartz, "On the complexity of approximating k-set packing," *computational complexity*, vol. 15, no. 1, pp. 20–39, 2006.

[31] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1863–1876, 2016.

[32] X. Xia, F. Chen, G. Cui, M. Abdelrazek, J. Grundy, H. Jin, and Q. He, "Budgeted data caching based on k-median in mobile edge computing," in *27th IEEE International Conference on Web Services*. IEEE, 2020, pp. 197–206.

[33] B. Li, Q. He, G. Cui, X. Xia, F. Chen, H. Jin, and Y. Yang, "Read: Robustness-oriented edge application deployment in edge computing environment," *IEEE Transactions on Services Computing*, 2020.

[34] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 483–498.

[35] B. Banerjee, A. Kulkarni, and A. Seetharam, "Greedy caching: An optimized content placement strategy for information-centric networks," *Computer Networks*, vol. 140, pp. 78–91, 2018.

[36] M. Y. S. Uddin and N. Venkatasubramanian, "Edge caching for enriched notifications delivery in big active data," in *38th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 696–705.

[37] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," in *33rd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2013, pp. 62–72.

[38] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "Cloudcache: On-demand flash cache management for cloud computing," in *14th USENIX Conference on File and Storage Technologies(FAST)*, 2016, pp. 355–369.

[39] K. Ma, B. Yang, Z. Yang, and Z. Yu, "Segment access-aware dynamic semantic cache in cloud computing environment," *Journal of Parallel and Distributed Computing*, vol. 110, pp. 42–51, 2017.

[40] A.-H. A. Badawy, G. Yessin, V. Narayana, D. Mayhew, and T. El-Ghazawi, "Optimizing thin client caches for mobile cloud computing: Design space exploration using genetic algorithms," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 11, pp. 1–13, 2017.

[41] S. Tamoor-ul Hassan, S. Samarakoon, M. Bennis, M. Latva-Aho, and C. S. Hong, "Learning-based caching in cloud-aided wireless networks," *IEEE Communications Letters*, vol. 22, no. 1, pp. 137–140, 2018.

[42] F. Chen, J. Zhou, X. Xia, H. Jin, and Q. He, "Optimal application deployment in mobile edge computing environment," in *13th IEEE International Conference on Cloud Computing*. IEEE, 2020, pp. 184–192.

[43] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing cache data integrity in the edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[44] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *38th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 388–399.

[45] R. Halalai, P. Felber, A.-M. Kermarrec, and F. Taïani, "Agar: A caching system for erasure-coded data," in *37th IEEE International Conference onDistributed Computing Systems (ICDCS)*, 2017, pp. 23–33.

[46] X. Zhang and Q. Zhu, "Hierarchical caching for statistical qos guaranteed multimedia transmissions over 5g edge computing mobile wireless networks," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 12–20, 2018.

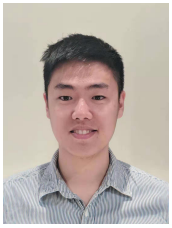
[47] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *37th IEEE International Conference onDistributed Computing Systems (ICDCS)*, 2017, pp. 276–286.

[48] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5g networks with mobile edge computing," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 80–87, 2018.

[49] X. Zhang and Q. Zhu, "Collaborative hierarchical caching over 5g edge computing mobile wireless networks," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[50] J. Zhou, J. Fan, J. Wang, and J. Jia, "Dynamic service deployment for budget-constrained mobile edge computing," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 24, pp. 1–16, 2019.

[51] S. Deng, Z. Xiang, J. Taheri, K. A. Mohammad, J. Yin, A. Zomaya, and S. Dustdar, "Optimal application deployment in resource constrained distributed edges," *IEEE Transactions on Mobile Computing*, 2020.



Xiaoyu Xia received his Master degree from The University of Melbourne, Australia in 2015. He is a PhD candidate at Deakin University. His research interests include edge computing, parallel and distributed computing, service computing, software engineering and cloud computing.

Feifei Chen received her PhD degree from Swinburne University of Technology, Australia in 2015. She is a lecturer at Deakin University. Her research interests include software engineering, cloud computing and green computing.

John C. Grundy received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently Australian Laureate Fellow and a professor of software engineering at Monash University, Melbourne, Australia. He is an associate editor of the IEEE Transactions on Software Engineering, the Automated Software Engineering Journal, and IEEE Software. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at https://sites.google.com/site/johncgrundy/.

Mohamed Abdelrazek is an Associate Professor of Software Engineering and IoT at Deakin University. Before joining Deakin University in 2015, he worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). Before 2010, he was the head of software development department at Microtech. More details about his research can be found at https://sites.google.com/site/mohamedalmorsy/.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. Jin received his PhD in computer engineering from HUST in 1994. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Qiang He received his first PhD degree from Swinburne University of Technology, Australia, in 2009 and his second PhD degree in computer science and engineering from Huazhong University of Science and Technology, China, in 2010. He is an Associate Professor at Swinburne. His research interests include service computing, software engineering, cloud computing and edge computing. More details about his research can be found at https://sites.google.com/site/heqiang/.

3.3 Summary

Chapter 3 makes the first attempt to solve data placement problems in edge computing from an app vendor's perspective, and answers **RQ1**. In this chapter, we first investigate the individual data placement problem in edge computing and propose two approaches, including an optimal approach and a greedy algorithm. After that, considering three major issues in **Chapter 3.1**, we study the multi-data placement problem under the reserved cache space constraint via multiple hops in the edge computing environment. Similar to **Chapter 3.1**, we propose an optimal solution to solve this problem exactly and an approximation algorithm to find solutions effectively and efficiently. This chapter has established the foundation for the edge data placement problem and opened up a number of research directions.

Chapter 4

Edge Data Replacement

In the edge computing environment, data should be placed and replaced dynamically over time. Placed data may need to be flushed out to save cache spaces for more popular data. Moreover, users arrive and leave the specific area covered by edge servers over time. These dynamic data demands and user mobility are not known in advance and must be accommodated on the fly. To address the dynamics in edge computing and answer **RQ2**, this chapter aims to investigate the edge data replacement problem from the app vendor's perspective, comprised of two published papers.

4.1 Lyapunov-Based Collaborative Data Replacement Strategies in Edge Computing

From the app vendor's perspective, it is critical to find a suitable data replacement strategy that minimizes the total cost with limited storage spaces on edge servers while fulfilling the unique constraints of edge computing, including server capacity constraint, server coverage constraint and server adjacency constraint. To optimally place and replace data on edge servers, complete information about the edge computing system overtime is required. However, it is unrealistically in real-world scenarios where users' data requests usually arrive dynamically. In this chapter, we propose a Lyapunov-based collaborative data replacement approach named CEDC-O to minimize the total cost while ensuring low data retrieval latency. We evaluate this approach on a real-world dataset, and the results demonstrate that our approach outperforms the state-of-the-art approaches significantly.

This chapter is based on a published paper, entitled: Online Collaborative Data Caching in Edge Computing, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32(2), pp. 281-294.

Online Collaborative Data Caching in Edge Computing

Xiaoyu Xia, Feifei Chen, Qiang He, *Senior Member, IEEE*, John Grundy, *Senior Member, IEEE*, Mohamed Abdelrazek, and Hai Jin, *Fellow, IEEE*

Abstract—In the edge computing (EC) environment, edge servers are deployed at base stations to offer highly accessible computing and storage resources to nearby app users. From the app vendor's perspective, caching data on edge servers can ensure low latency in app users' retrieval of app data. However, an edge server normally owns limited resources due to its limited size. In this article, we investigate the *collaborative caching problem* in the EC environment with the aim to minimize the system cost including data caching cost, data migration cost, and quality-of-service (QoS) penalty. We model this *collaborative edge data caching problem* (CEDC) as a constrained optimization problem and prove that it is $\mathcal{NP}$-complete. We propose an online algorithm, called CEDC-O, to solve this CEDC problem during all time slots. CEDC-O is developed based on Lyapunov optimization, works online without requiring future information, and achieves provable close-to-optimal performance. CEDC-O is evaluated on a real-world data set, and the results demonstrate that it significantly outperforms four representative approaches.

Index Terms—Edge computing, data caching, online algorithm

1 Introduction

The world has witnessed an exponential growth of mobile devices including mobile phones, wearable devices, tablets, smart vehicle and Internet-of-Things (IoT) devices over the last decade [1]. The enormous network traffic often causes network congestion and increases network latency. To address this issue, edge computing (EC), a new computing paradigm, has emerged to distribute computing capacities from centralized cloud to distributed edge servers [2]. Each edge server is powered by one or more physical devices and is attached to a base station or an access point that is geographically close to app users' mobile devices. Mobile and IoT application vendors (referred to as *app vendor* hereafter) can host their apps on edge servers (referred to as *edge apps* hereafter) to ensure low latency and high-quality services for their app users by hiring computing and storage resources on edge servers [3]. Computation tasks can be offloaded from mobile devices to nearby edge servers to reduce the computation overhead and energy consumption on those mobile devices [4], [5], [6], [7]. This is a key technology that facilitates the 5G mobile network [8].

As a rapidly increasing number of app users begin to access edge apps, more mobile data will be transmitted through edge servers between the cloud and app users' mobile devices. From an app vendor's perspective, caching those data, especially popular ones like viral videos and posts from Facebook[1] and Twitter,[2] will significantly reduce network delay in app users' retrieval of app data. App users can retrieve data from nearby edge servers instead of from the remote cloud servers if the data are already cached on those edge servers. In addition, caching data on edge servers can also considerably reduce the amount of data transferred between the cloud and the mobile devices, which consequently lower app vendors' cost of data transfer under the pay-as-you-go pricing scheme.

Data caching techniques have been widely implemented in many different domains, from hardware cache, e.g., CPU [9], GPU [10], memory [11], disks [12], to software cache, e.g., web [13], database [14], etc. In the network domain, data caching has also been intensively studied to leverage its advantages in saving bandwidth consumption, reducing network latency and minimizing access costs [15], [16], [17]. In the last few years, many researchers have investigated network cache from different perspectives, e.g., cache allocation and replacement strategies [18], coded caching [19], request routing [20], and information-theoretic caching [21], [22]. As a new computing paradigm, EC offers new opportunities and raises new challenges for data caching. The fundamental objective and mechanism are to cache popular data on edge servers so that nearby app users can retrieve the cached data with low latency. This is especially

- *Xiaoyu Xia, Feifei Chen, and Mohamed Abdelrazek are with the School of Information Technology, Deakin University, Geelong, Victoria 3220, Australia. E-mail: {xiaoyu.xia, feifei.chen, mohamed.abdelrazek}@deakin.edu.au.*
- *Qiang He is with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria 3122, Australia. E-mail: qhe@swin.edu.au.*
- *John Grundy is with the Faculty of Information Technology, Monash University, Melbourne, Victoria 3800, Australia. E-mail: john.grundy@monash.edu.*
- *Hai Jin is with the Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technolgoy, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: hjin@hust.edu.cn.*

Manuscript received 16 Jan. 2020; revised 17 June 2020; accepted 3 Aug. 2020. Date of publication 13 Aug. 2020; date of current version 24 Aug. 2020. (Corresponding author: Qiang He.) Recommended for acceptance by Y. Yang. Digital Object Identifier no. 10.1109/TPDS.2020.3016344

1. https://www.facebook.com/
2. https://www.twitch.tv/

important for latency-sensitive applications, e.g., interactive gaming, real-time navigation, augmented reality, etc. In addition, caching data on edge servers can also lift the traffic burden on the Internet backbone by reducing the amount of mobile traffic data transmitted between the cloud and app users' mobile devices significantly [23].

Given a set of popular data, from an app vendor's perspective, a straightforward solution is to cache them all on every edge server in a particular area to minimize the latency in its app users' data retrieval in that area. In addition to data latency, the app vendor also needs to consider the cost of hiring storage resources on edge servers for data caching based on the pay-as-you-go pricing model. The cost also occurs during data transmission and migration among the network. Thus, from an app vendor's perspective, it is critical to find a collaborative data caching strategy that minimizes the total system cost with limited storage spaces on edge servers while fulfilling the above mentioned constraints in the edge computing environment, including server capacity constraint, server coverage constraint and server adjacency constraint. Over time, a lot of data will be cached on the edge servers and new data will replace old data. An app vendor's hired storage resources on edge servers and its cached app data constitute an edge caching system. In the long-term, how to keep an app vendor's edge caching system stable over time across multiple time slots is the key problem to be solved in this paper.

We refer to this data caching problem in the EC environment *collaborative edge data caching* (CEDC) problem. To the best of our knowledge, this work is the first attempt to solve this CEDC problem from the app vendor's perspective. The key contributions of this work are as follows:

- We model and formulate the CEDC problem as a constrained optimization problem from the app vendor's perspective.
- We prove that the CEDC problem is $\mathcal{NP}$-complete.
- We propose an online algorithm named CEDC-O based on Lyapunov optimization to solve the CEDC problem across multiple time slots without requiring future information, and prove the performance bounds of this algorithm.
- We evaluate the performance of our algorithm by extensive simulations conducted on a real-world data set.

The rest of the paper is organized as follows. Section 2 presents an example to illustrate and motivate the CEDC problem. Section 3 presents the system model, formulates the CEDC problem and proves its $\mathcal{NP}$-completeness. Section 4 presents the CEDC-O algorithm and theoretically analyzes its performance bounds. Section 5 evaluates the CEDC-O algorithm experimentally. Section 6 reviews the related work, followed by the conclusion in Section 7.

2 Motivating Example

EC is significantly different from cloud computing which facilitates the content-centric network and the content delivery network. In the EC environment, adjacent edge servers deployed at different base stations can communicate with their neighbor edge servers and transmit data via high-speed links [4], [24]. App users' workloads in a particular area can be transferred and balanced across the edge servers covering that area [4]. This architecture overcomes the single-point failure problem encountered by the one with the macro base station [25]. Thus, the edge servers in a particular area can constitute a graph, namely *edge server network*, where a node represents an edge server and an edge represents the link between two edge servers.

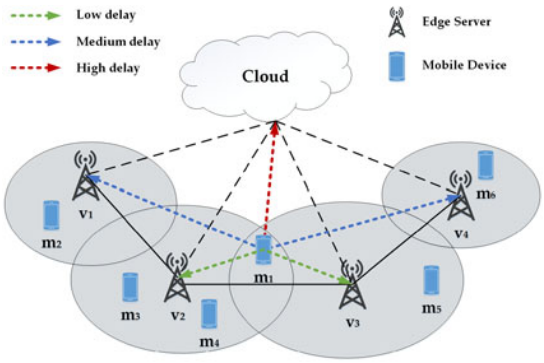


Fig. 1. An example scenario for edge computing.

Fig. 1 shows an example of a typical edge environment involving a set of mobile devices, $\{m_1, \ldots, m_6\}$, and edge servers, $\{v_1, \ldots, v_4\}$. Mobile devices connect to edge servers to retrieve data. Compared to cloud servers, the storage resources on an edge server are usually limited due to their limited sizes [26]. This intensifies the competition between app vendors for computing and storage resources on edge servers, making it extremely expensive and, in most cases impossible, for an app vendor to cache all its app data on every edge server. In such an environment, the common practice for an app vendor is to reserve a number of cache spaces on each edge servers for caching its most popular data. This is a fundamental difference between cloud computing and EC because the computing and storage resources available in the cloud are often assumed to be virtually unlimited. The limited resources on edge servers are referred to as the *server capacity constraint*. Furthermore, data caching in the EC environment differs from data caching in the cloud computing environment as well as other conventional distributed computing environments with its two unique constraints, i.e., *server coverage constraint* and *server adjacency constraint*:

Server coverage constraint: An edge server covers a specific geographical area so that app users' mobile devices within its coverage area can connect to it via LTE or Radio Network [27]. In a particular area, a number of edge servers are usually deployed in a distributed manner so that they can cover different geographical areas. The coverage areas of adjacent edge servers usually partially overlap to avoid blank areas not covered by any edge servers. For example, mobile device m_1 in Fig. 1 can directly access edge server v_2 and v_3 while m_2 can only access edge server v_1 directly.

Server adjacency constraint: An app user can retrieve cached app data from its nearby edge servers (referred to as *local edge servers* hereafter) if the data is cached on any of these edge servers. Otherwise, the data can be retrieved from those local edge servers' *neighbor edge servers*, i.e., edge servers that are directly linked to them via high-speed links [24]. Either way, it is faster than retrieving the data from a remote

cloud server [24]. Take m_1 in Fig. 1 as an example. This device can access the caches in its local edge servers v_2 and v_3, or its neighbor edge servers v_1 and v_4, or the remote cloud. The only difference is the data retrieval latency, which is represented by the different colors of the links in Fig. 1.

3 System Model

In this section, we first introduce the system architecture for edge data caching, then define the three components of system cost, including data caching cost, data migration cost and QoS penalty based on the constrains discussed in Section 2. The notations adopted in this section are summarized in Table 1.

3.1 System Architecture

In this research, we model the edge server network within a specific area as a graph $G(V, E)$ where $V = \{v_1, \ldots, v_n\}$ is the set of nodes and $E = \{e_1, \ldots, e_k\}$ is the set of edges in G. In this graph, each node $v \in V$ represents an edge server, while each edge $e \in E$ represents the link between two edge servers in G. *In the remainder of this paper, we will speak interchangeably of an edge server and its corresponding node in G, both denoted by v.*

To quantify the optimization objective and constraints in the CEDC problem in a generic manner, we measure the data sizes and cache spaces by the number of data units, and the data retrieval latency with the number of hops. Take Fig. 1 as an example. The cost of caching data d on all the four edge servers is 4. When data d is only cached on edge server v_3, Device 1 can retrieve the data from its local edge server v_3 via 0 hop, while Device 1 can retrieve the data from its neighbor edge server v_4 via 1 hop. This way, these models can be easily extended by integrating specific pricing models and latency models from edge infrastructure providers.

Given a set of data D required by app users in a specific area in time slot t, a data caching strategy to cover those data requests can be presented as $\lambda^t = \{\lambda_1^t, \ldots, \lambda_n^t\}$, where $\lambda_i^t = \{\lambda_{i,d}^t, \forall d \in D\}$. $\lambda_{i,d}^t$ denotes whether data d is cached on edge server v_i:

$$\lambda_{i,d}^t = \begin{cases} 0 & \text{if data } d \text{ is not cached on } v_i \text{ in time slot } t \\ 1 & \text{if data } d \text{ is cached on } v_i \text{ in time slot } t \end{cases}. \tag{1}$$

Let us denote $\tau_{m,d}^t$ as whether the request for data d from the app user's mobile device m exists in time slot t:

$$\tau_{m,d}^t = \begin{cases} 0 & \text{if the } m\text{'s request for data } d \text{ does} \\ & \text{not exist in time slot } t \\ 1 & \text{if the } m\text{'s request for data } d \text{ exists in } t \end{cases}. \tag{2}$$

Since the data requests arrive randomly in the stochastic EC environment, we model the data request arrivals as an independent and identical distribution, similar to many studies in the fields of edge computing, cloud computing and wireless networking [28], [29], [30].

TABLE 1
Notations in Our System Model

Notation	Description
A_i	available cache spaces on edge server i
$\mathcal{C}(\lambda^t)$	total system cost in time slot t
$\mathcal{C}_\mathcal{D}(\lambda^t)$	cost of data storage in time slot t
$\mathcal{C}_\mathcal{M}(\lambda^t)$	cost of data migration in time slot t
$\mathcal{C}_\mathcal{P}(\lambda^t)$	QoS penalty in time slot t
c_l	unit cost of data latency
c_{mc}	unit cost of data migration from cloud
c_{ms}	unit cost of data migration from edge server
c_s	unit cost of data storage
d	requested data d
D	finite set of requested data
E	finite set of links between edge servers
G	graph presenting the edge server network
$l_{i,j}$	hops between edge server v_i and v_j
$l_{i,d}^t$	lowest latency to migrate data d for v_i from the edge server network in time slot t
$l_{m,d}^t$	lowest latency of for mobile m to retrieve d from the edge server network in time slot t
$l_\mathcal{T}$	latency limit accepted by the app vendor
$\mathcal{L}$	long-term average latency constraint
m	mobile device m
M	set of mobile devices
M_j	set of mobile devices covered by server v_j
t	time slot t
T	infinite set of time
V	set of edge servers
v_i	edge server i
$\mathcal{X}_{i,d}^t$	binary variable indicating whether data d has been already cached on edge server v_i at the beginning of time t
$\mathcal{Y}_{i,d}^t$	binary variable indicating whether edge server i can retrieve data d from a neighbor edge server or the remote cloud
λ^t	data caching strategy in time slot t
$\lambda_{i,d}^t$	binary variable indicating whether data d will be cached on edge server v_i at the end of time t
$\tau_{m,d}^t$	binary variable indicating whether the mobile device m requests for data d in time slot t
ρ	ratio of c_{mc} over c_s
η	ratio of c_{ms} over c_s
ω	ratio of c_p over c_s

As mentioned in Section 1, the storage resources on an edge server is usually limited. Thus, the competition between app vendors usually makes it impossible for an app vendor to cache all its app data on every edge server. Thus, the number of data cached in any time slot t on each edge server v_i cannot violate the available server capacity constraint:

$$\sum_{d \in D} \lambda_{i,d}^t \leq A_i, \forall t = \{0, ..., T-1\}, i = \{1, \ldots, n\}. \tag{3}$$

3.2 Data Retrieval Latency

The data retrieval latency in the edge server network consists of two components: the latency between the device and its nearby edge server, and the latency between its local

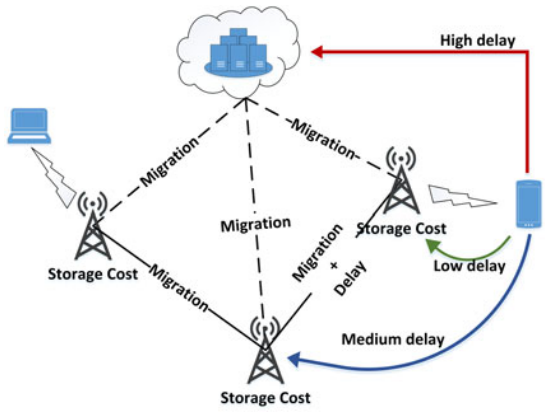


Fig. 2. System Cost: Data caching cost, migration cost and QoS penalty.

edge server and neighbor edge server. As the first component is extremely small in the 5G network and not influenced by the data caching strategy, it is not considered in the formulation of the data caching strategy. Thus the network delay in retrieving data d for the app user's mobile device m in time slot t is calculated as follows:

$$l^t_{m,d} = \min\{l_{i,j}, \lambda^t_{i,d} = 1, v_i \in V\}, \forall m \in M_j, \tag{4}$$

where v_i is the edge server caching data d and v_j is the edge server covering the app user's mobile device m, and $l_{i,j}$ is the number of hops between v_i and v_j.

We denote $l_{\mathcal{T}}$ as the latency limit specified by the app vendor. If the latency $l^t_{m,d}$ is higher than $l_{\mathcal{T}}$, mobile device m will access the data directly from the remote cloud. In this way, the delay in m's retrieval of data d from the remote cloud server c in time slot t is calculated as $l^t_{m,d} = l_{c,m}$, where $l_{c,m} > l_{\mathcal{T}}$ is the latency between c and m.

3.3 System Cost

From the app vendor's perspective, a key performance indicator for its edge caching system is the total system cost produced by the data caching strategy.

Fig. 2 depicts the key elements of the system cost model. *Data caching cost* is measured based on the storage resources hired by the app vendor in each time slot. *Data migration cost* is produced by migrating data from the cloud or the neighbor edge servers to the local edge servers. *QoS penalty* is the third component of the system cost, occurring when a user has to retrieve data from the cloud server with a high latency.

3.3.1 Data Caching Cost

It is calculated by how many cache spaces hired by the app vendor. As mentioned above, the cache spaces are measured by the number of data units. Thus, the data caching cost in time slot t can be calculated as follows:

$$\mathcal{C}_{\mathcal{D}}(\lambda^t) = \sum_{v_i \in V} \sum_{d \in D} c_s \lambda^t_{i,d}, \tag{5}$$

where c_s is the unit cost of hiring data resources on an edge server for data caching.

3.3.2 Data Migration Cost

As transferring new data from the remote cloud to an edge server or between edge servers to be cached incurs additional network delay, data migration cost is incurred and is calculated based on the number of new cached data. Here, we denote $\mathcal{X}^t_{i,d} = 0$ if data d is already cached on edge server v_i at the start of time slot t, otherwise $\mathcal{X}^t_{i,d} = 1$:

$$\mathcal{X}^t_{i,d} = 1 - \lambda^{t-1}_{i,d}. \tag{6}$$

Similar to (4), we denote $l^t_{i,d} = min\{l_{i,j}, \lambda^{t-1}_{j,d} = 1, \forall v_j \in V\}$ as the lowest latency for edge server v_i to obtain data d in time slot t, over the edge server network. We denote the unit cost of data migration from the cloud to the app user's mobile device by c_{mc}, and the unit cost of data transmission from a neighbor edge server to mobile device by c_{ms}. If the cost of data migration over the edge server network is higher than that from the remote cloud ($c_{ms} \cdot l^t_{i,d} > c_{mc}$), v_i will retrieve the data from the remote cloud directly. Here we denote the source of a requested data as $\mathcal{Y}^t_{i,d} \in \{0, 1\}$:

$$\mathcal{Y}^t_{i,d} = \begin{cases} 0 & \text{if } c_{ms} \cdot l^t_{i,d} > c_{mc} \\ 1 & \text{if } c_{ms} \cdot l^t_{i,d} \leq c_{mc} \end{cases}. \tag{7}$$

Thus, we obtain the data migration cost as follows:

$$\mathcal{C}_{\mathcal{M}}(\lambda^t) = \sum_{v_i \in V} \sum_{d \in D} \lambda^t_{i,d} \mathcal{X}^t_{i,d} ((c_{ms} \cdot l^t_{i,d} - c_{mc}) \mathcal{Y}^t_{i,d} + c_{mc}). \tag{8}$$

We denote ρ as the ratio of c_{mc} over c_s and η as the ratio of c_{ms} over c_s, then we obtain:

$$\mathcal{C}_{\mathcal{M}}(\lambda^t) = \sum_{v_i \in V} \sum_{d \in D} c_s \lambda^t_{i,d} \mathcal{X}^t_{i,d} ((\eta \cdot l^t_{i,d} - \rho) \mathcal{Y}^t_{i,d} + \rho). \tag{9}$$

3.3.3 QoS Penalty

As mentioned in Section 2, the data can be retrieved from local edge servers or neighbor edge servers. Either way, it is faster than retrieving the data from a remote cloud server [24]. Thus, the quality-of-service (QoS) is impacted significantly for users who cannot retrieve data from any available edge servers within $l_{\mathcal{T}}$. Thus, the QoS penalty occurs when a user has to retrieve data from the remote cloud server or from an edge server with a limit-violating latency. Here, we denote $\theta^t_{m,d} \in \{0, 1\}$ to indicate whether a QoS penalty is applied to m's retrieval of data d:

$$\theta^t_{m,d} = \begin{cases} 1 & \text{if } l^t_{m,d} > l_{\mathcal{T}} \\ 0 & \text{if } l^t_{m,d} \leq l_{\mathcal{T}} \end{cases}. \tag{10}$$

Please notice that $l_{c,m} > l_{\mathcal{T}}, \forall m \in M$. This way, $\theta^t_{m,d}$ is always 1.

We denote c_p as the unit cost incurred by QoS penalty, determined by the app vendor based on its priority for its app users' QoS. The QoS penalty in time slot t, as part of system cost, is calculated as:

$$\mathcal{C}_{\mathcal{P}}(\lambda^t) = \sum_{m \in M} \sum_{d \in D} c_p \theta^t_{m,d} = \sum_{m \in M} \sum_{d \in D} \omega \cdot c_s \theta^t_{m,d}, \tag{11}$$

where ω is the ratio of c_p over c_s.

3.4 Problem Formulation and Hardness

With the consideration of the system architecture in Section 3.1 and the costs presented in Section 3.3, the total system cost is calculated by summing all the aforementioned costs:

$$\begin{aligned}\mathcal{C}(\lambda^t) &= \mathcal{C}_{\mathcal{D}}(\lambda^t) + \mathcal{C}_{\mathcal{M}}(\lambda^t) + \mathcal{C}_{\mathcal{P}}(\lambda^t) \\ &= c_s\Big(\sum_{v_i\in V}\sum_{d\in D}\lambda^t_{i,d}\big(1+\mathcal{X}^t_{i,d}((\eta-\rho)\mathcal{Y}^t_{i,d}+\rho)\big)\Big) \\ &\quad + \sum_{m\in M}\sum_{d\in D}\omega\cdot c_s\theta^t_{m,d}.\end{aligned} \tag{12}$$

In a CEDC scenario, from the app vendor's perspective, it is important to minimize the system cost incurred by caching data on edge servers, migrating data across edge servers and failing to serve users on edge servers. While pursing this optimization objective, it is also necessary to stabilize the time-averaged system latency perceived by the users in the long term. Thus, an app vendor usually has a long-term average system latency constraint for requests served by edge servers, denoted by $\mathcal{L}$. Thus, the following inequality must be fulfilled:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\frac{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}\cdot l^t_{m,d}}{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}} \le \mathcal{L}. \tag{13}$$

In the stochastic EC environment, data requests randomly arrive and leave [28]. Thus, the app vendor's long-term system performance usually outweighs its immediate short-term system performance. Thus, we formulate the CEDC problem over multiple time slots as a constrained optimization problem (COP):

$$\begin{aligned}\mathcal{P}_1 :& \min \lim_{T\to\infty}\sum_{t=0}^{T-1}\mathcal{C}(\lambda^t) \\ s.t. :& (1),(3),(4),(6),(13).\end{aligned}$$

Now, we demonstrate that the COP of CEDC problem in a single time slot t is $\mathcal{NP}$-complete by proving the following theorems.

Theorem 1. *The COP of CEDC in time slot t is $\mathcal{NP}$.*

Proof. As there are $(1+|M|+2|V|)|D|$ constraints in total, any solution to the COP can be validated in polynomial time by checking whether the solution satisfies the constraints (1), (3), (6) and (4). Thus, the CEDC problem is $\mathcal{NP}$.□

Theorem 2. *The COP of CEDC in time slot t is $\mathcal{NP}$-complete.*

Proof. To prove this problem is $\mathcal{NP}$-complete, we first introduce the weighted k-set packing problem. The weighted k-set packing problem is known to be $\mathcal{NP}$-complete [31]. Given a universe U with elements $\forall e \in U$, a set U' of subsets of U and an integer number k. The subset X is a packing, where $X \subseteq U'$. All sets $x \subseteq X$ are pairwise disjoint. Let $\mathcal{W}(x)$ be the weight of the set x and k be the maximum number of selected sets. The formulation is displayed below:

$$object : \max \sum_{x\in X}\mathcal{W}(x)\mathcal{T}_x \tag{14a}$$

$$s.t. : \sum_{x\in X}\mathcal{T}_x \le k \tag{14b}$$

$$\mathcal{T}_x \in \{0,1\}, \forall x \in X \tag{14c}$$

$$\sum_{e\in U}\mathcal{T}_e \le 1. \tag{14d}$$

Next, we prove that the weighted k-set packing problem can be reduced to an instance of the COP of CEDC problem. We define the elements based on the data requests from app user's mobile devices. The reduction can be done as follows: 1) adding the cloud server v_{cloud} as a node into the graph; 2) adding edges from v_{cloud} to all other nodes in the graph; 3) setting the storage cost incurred on v_{cloud} to 0; 4) setting $l_{\mathcal{T}} = \mathcal{L} = |V|$. After the above reduction, constraint (13) can be ignored. Given any instance $WeightedKSet(X, U, k, \mathcal{W}(x))$, we can construct $CEDC(V, M, n, Benefits(i, d))$ with the reduction above in polynomial time while $|X| = |V|$, $|U| = |M|$ and $n = k$. The function $Benefits(i, d)$ is calculated as the reduced QoS penalty minus the data caching cost if data d is cached on edge server v_i. As the constraint (14b) restricts the total number of selected sets, we can project (3) in the CEDC problem to that constraint. Based on (4), the latency in mobile device m's retrieval of data d is the lowest latency between m and any edge server with d in the cache. Thus, constraint (14d) can be fulfilled. Moreover, we can convert our original objective (12) to $\mathcal{C}_{\mathcal{D}}(\lambda^t) + \mathcal{C}_{\mathcal{L}}(v_{cloud}) - \sum_{v_i\in V}\sum_{d\in D}\lambda^t_{i,d}Benefits(i, d)$. Then, we can project our converted objective to the objective (14a). Thus, the COP of CEDC problem in time slot t is reducible from the weighted k-set packing problem, and it is $\mathcal{NP}$-complete. □

4 Online Caching Algorithm Design

To solve the CEDC problem optimally, the complete information about the system over all the time slots must be known. However, this cannot be realistically fulfilled for real-world scenarios. To practically fulfil the app vendor's long-term latency constraint (13), we need to convert $\mathcal{P}_1$, a non-convex problem, to a linear and convex problem. To do so, we propose an Online Collaborative Edge Data Caching (CEDC-O) algorithm based on Lyapunov optimization [32] for finding near-optimal solutions to the CEDC problem in individual time slots without future information. The notations adopted in this section are summarized in Table 2.

4.1 Online Collaborative Edge Data Caching Algorithm

We provide an online algorithm, named CEDC-O, based on Lyapunov optimization, to convert the long-term optimization problem $\mathcal{P}_2$ to optimization problems in individual time slots. The most significant characteristic of CEDC-O is that it only requires the information in the current time slot rather than the complete information in all the time slots

TABLE 2
Notations in Our Algorithm Design

Notation	Description
$\mathcal{C}$	value of system cost produced by λ
$\mathcal{C}'$	value of system cost produced by λ'
$\mathcal{C}_{opt}$	value of system cost produced by λ_{opt}
$\mathcal{C}_{min}$	smallest system cost of all possible solutions
$\mathcal{C}_{max}$	largest system cost of all possible solutions
$\mathcal{DP}(t)$	Lyapunov drift-plus-penalty function
$\mathcal{L}_{avg}(\lambda^t)$	average system latency in time slot t
$\mathcal{L}(\sigma(t))$	Lyapunov function, calculated by $\frac{1}{2}\sigma^2(t)$
$\mathcal{Q}$	constant value equal to $\frac{1}{2}\mathcal{L}^2$
$\mathcal{Q}'$	constant value equal to $\mathcal{Q}+\gamma\cdot(\mathcal{C}_{max}-\mathcal{C}_{min})$
γ	positive parameter adjusting the trade-off between system cost $\mathcal{C}(\lambda^t)$ and the average latency $\mathcal{L}_{avg}(\lambda^t)$
λ	solution obtained by CEDC-O
λ^t	λ in time slot t
λ'	feasible solution fulfilling (21)
λ'^t	λ' in time slot t
λ^*	feasible solution fulfilling (27)
λ^{*t}	λ^* in time slot t
λ_{opt}	optimal solution to $\mathcal{P}_1$ over all time slots
λ^t_{opt}	λ_{opt} in time slot t
$\sigma(t)$	accumulated latency in time slot t
$\Delta(\sigma(t))$	Lyapunov drift function

when solving $\mathcal{P}_2$. While trying to minimize the system cost, the app vendor also needs to stabilize the system latency to ensure low-latency data access for its users. Thus, in this paper, the system metric to stabilize by CEDC-O is the time-averaged system latency perceived by the users over the long term.

Lyapunov optimization is typically applied in the communication and queuing systems. With the application of Lyapunov optimization, the problems can be formulated as problems that optimize the time averages of certain objectives subject to some time average constraints, and they can be solved with a common mathematical framework that is intimately connected to queuing theory [32]. Unlike the typical application of Lyapunov optimization that models the problem as a queuing network, we define the *accumulated latency* in Definition 1 to stabilize the system latency over time.

Definition 1 (Accumulated Latency). *Accumulated latency σ_t is the overdue delay accumulated over t time slots, calculated as:*

$$\sigma(t+1)=\max\{\sigma(t)+\mathcal{L}_{avg}(\lambda^t)-\mathcal{L},0\} \tag{15}$$

where $\mathcal{L}_{avg}(\lambda^t)=\frac{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}\cdot l^t_{m,d}}{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}}$, *and* $\sigma(0)=0$ *because there is no latency at the very beginning.*

Based on Definition 1, the accumulated latency will increase if the latency is over $\mathcal{L}$ in the previous time slot. This can be employed as a penalty to adjust the data caching strategy to stabilize the system latency over time as specified by (13). Now, we can convert the long-term latency constraint (13) to a new constraint based on accumulated latency:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\sigma(t)]\le 0. \tag{16}$$

Given (15), a Lyapunov function can be defined as $\mathcal{L}(\sigma(t))\triangleq\frac{1}{2}\sigma^2(t)$. It indicates the system stability measured by its accumulated latency $L(\sigma(t))$. Here, the Lyapunov drift $\Delta(\sigma(t))$ is applied in each time slot to enhance the system stability:

$$\begin{aligned}\Delta(\sigma(t))&=\mathbb{E}[\mathcal{L}(t+1)-\mathcal{L}(t)|\sigma(t)]\\&=\frac{1}{2}\mathbb{E}[\sigma^2(t+1)-\sigma^2(t)|\sigma(t)]\\&=\frac{1}{2}\mathbb{E}[(\mathcal{L}_{avg}(\lambda^t)-\mathcal{L})^2|\sigma(t)]+\sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^t)-\mathcal{L}|\sigma(t)]\\&\le\mathcal{Q}+\sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^t)-\mathcal{L}|\sigma(t)],\end{aligned} \tag{17}$$

where $\mathcal{Q}=\frac{1}{2}\mathcal{L}^2$ because of $\mathcal{L}_{avg}(\lambda^t)\ge 0$.

As we obtain the upper bound of the Lyapunov drift function, we introduce the penalty in our CEDC-O algorithm based on the total cost objective (12). We denote γ as a positive parameter in Lyapunov optimization for adjusting the trade-off between the system cost $\mathcal{C}(\lambda^t)$ and the number of time slots needed to converge the time-averaged latency back to $\mathcal{L}$ when (13) is violated. Here, we introduce the Lyapunov *drift-plus-penalty* function $\mathcal{DP}(t)$, defined as:

$$\mathcal{DP}(t)=\Delta(\sigma(t))+\gamma\cdot\mathbb{E}[\mathcal{C}(\lambda^t)|\sigma(t)]. \tag{18}$$

In each time slot, the data caching strategy is formulated to minimize the total cost $\mathcal{C}(\lambda^t)$ and to keep the system stable, and we can get the upper bound of this function by:

$$\mathcal{DP}(t)\le\mathcal{Q}+\sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^t)-\mathcal{L}|\sigma(t)]+\gamma\cdot\mathbb{E}[\mathcal{C}(\lambda^t)|\sigma(t)]. \tag{19}$$

The pseudocode of the CEDC-O algorithm is presented in Algorithm 1. In each time slot, the data caching strategy is formulated by finding the optimal solution to $\mathcal{P}_2$:

$$\begin{aligned}&\mathcal{P}_2:\min(\mathcal{Q}+\sigma(t)(\mathcal{L}_{avg}(\lambda^t)-\mathcal{L})+\gamma\cdot\mathcal{C}(\lambda^t))\\&s.t.:(1),(3),(6),(4),(15).\end{aligned}$$

Algorithm 1. CEDC-O Algorithm

1: **Input:** $G=(V,E),M,A=\{A_1,\dots,A_n\},\mathcal{L},c_s,\rho,\eta,\omega$
2: **Output:** data caching strategy $\lambda=\{\lambda^1,\dots,\lambda^T\}$
3: $\sigma(0)=0,t=0$
4: **repeat**
5: Observe the data requests $\tau^t=\{\tau^t_{m,d}|\forall m\in M,d\in D\}$
6: Find the solution λ^t, where:

$$\lambda^t=\arg\min(\mathcal{Q}+\sigma(t)(\mathcal{L}_{avg}(\lambda^t)-\mathcal{L})+\gamma\cdot\mathcal{C}(\lambda^t)) \tag{20}$$

7: Observe $\mathcal{L}_{avg}(\lambda^t)$ and update $\sigma(t)$ based on (15)
8: $t=t+1$
9: **until** $t=T$

In Algorithm 1, no further information is required to solve $\mathcal{P}_2$ except the data requests in the current time slot. After implementing the *drift-plus-penalty* function, our CEDC-O algorithm considers the additional term $\sigma(t)(\mathcal{L}_{avg}(\lambda^t)-\mathcal{L})$, while $\mathcal{Q}$ is a constant. This drift-plus-penalty helps stabilize the system's average latency $\mathcal{L}_{avg}(\lambda^t)$ around

$\mathcal{L}$. Once $\mathcal{L}_{avg}(\lambda^t)$ exceeds $\mathcal{L}$, a penalty is applied to $\mathcal{P}_2$ and drives the CEDC-O algorithm to lower the system latency. Moreover, when $\sigma(t)$ increases, minimizing $\mathcal{L}_{avg}(\lambda^t)$ is of high significance in stabilizing the system and converging $\mathcal{L}_{avg}(\lambda^t)$ to the long-term budget (16). This is validated experimentally in Section 5.

4.2 Performance Analysis

Now, we analyze the performance of the CEDC-O algorithm based on the following theorems.

Theorem 3. *The time-averaged system cost of CEDC-O algorithm is bounded by $O(\frac{1}{\gamma})$.*

Proof. Let us assume that the optimal solution to $\mathcal{P}_1$ is $\lambda_{opt} = \{\lambda^0_{opt}, \ldots, \lambda^{T-1}_{opt}\}$. The average system cost of λ_{opt} is $\mathcal{C}_{opt} = \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)]$. The following inequality has been proven in [32]:

$$\exists \lambda'^t, \mathbb{E}[\mathcal{L}_{avg}(\lambda'^t) - \mathcal{L}|\sigma(t)] \leq \theta, \theta \to 0^+. \tag{21}$$

The CEDC-O algorithm provides the solution that minimizes $\mathcal{P}_2$ from all feasible decisions including λ' which contains λ'^t. Thus, we can obtain:

$$E[\mathcal{C}(\lambda'^t)|\sigma(t)] \leq E[\mathcal{C}(\lambda^t_{opt})|\sigma(t)]. \tag{22}$$

After applying (21) to (19), we can obtain:

$$\begin{aligned}
&\Delta(\sigma(t)) + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)] \\
&\overset{(19)}{\leq} \sigma(t)\mathbb{E}[(\mathcal{L}_{avg}(\lambda'^t) - \mathcal{L})|\sigma(t)] + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)] + \mathcal{Q} \\
&\overset{(22)}{\leq} \sigma(t)\mathbb{E}[(\mathcal{L}_{avg}(\lambda'^t) - \mathcal{L})|\sigma(t)] + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + \mathcal{Q} \\
&\overset{(21)}{\leq} \theta \cdot \sigma(t) + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + \mathcal{Q} \\
&= \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + \mathcal{Q}.
\end{aligned} \tag{23}$$

Based on (17) and (23), we sum all the $\Delta(\sigma(t))$ for all time slots and get:

$$\begin{aligned}
&\mathbb{E}[L(\sigma(T)) - L(\sigma(0))] + \gamma \cdot \sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)] \\
&\leq \gamma \cdot \sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + T \cdot \mathcal{Q} = \gamma \cdot T \cdot \mathcal{C}_{opt} + T \cdot \mathcal{Q}.
\end{aligned} \tag{24}$$

Denote $\mathcal{C}'$ as the value of system cost incurred by data caching strategy λ'. Considering the facts that $L(\sigma(T)) \geq 0$ and $L(\sigma(0)) = 0$, we can obtain:

$$\begin{aligned}
\mathcal{C}' &\leq \frac{1}{T}(\mathbb{E}[L(\sigma(T)) - L(\sigma(0))] + \lim_{T\to\infty}\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)]) \\
&\leq \mathcal{C}_{opt} + \frac{\mathcal{Q}}{\gamma}.
\end{aligned} \tag{25}$$

Based on (20), the performance $\mathcal{C}$ of solution λ provided by our CEDC-O algorithm always outperforms that of λ'. The distance of the time-averaged system cost between λ and λ_{opt} is:

$$\mathcal{C} - \mathcal{C}_{opt} \leq \mathcal{C}' - \mathcal{C}_{opt} \leq \frac{\mathcal{Q}}{\gamma}. \tag{26}$$

Thus, the time-average system cost of our CEDC-O algorithm is bounded by $O(\frac{\mathcal{Q}}{\gamma}) = O(\frac{1}{\gamma})$. □

Theorem 4. *By applying the CEDC-O algorithm, the time-average accumulated latency is bounded by $O(\gamma)$.*

Proof. Based on (15) and (16), we assume that there exist λ^{*t} and a positive value δ to fulfill:

$$\mathbb{E}[\mathcal{L}_{avg}(\lambda^{*t}) - \mathcal{L}|\sigma(t)] \leq -\delta. \tag{27}$$

Denote $\mathcal{C}_{min}$ and $\mathcal{C}_{max}$ as the smallest and largest system cost respectively. From (19), we obtain:

$$\begin{aligned}
\Delta(\sigma(t)) + \gamma \cdot \mathcal{C}_{min} \leq \mathcal{Q} + \gamma \cdot \mathcal{C}_{max} \\
+ \sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^{*t}) - \mathcal{L}|\sigma(t)]
\end{aligned}. \tag{28}$$

Define $Q' = Q + \gamma \cdot (\mathcal{C}_{max} - \mathcal{C}_{min})$. We have the following:

$$\begin{aligned}
\Delta(\sigma(t)) &\leq \mathcal{Q}' + \sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^{*t}) - \mathcal{L}|\sigma(t)] \\
&\overset{(27)}{\leq} \mathcal{Q}' - \delta \cdot \sigma(t).
\end{aligned} \tag{29}$$

By adding expectation to both sides of (29), we obtain:

$$\mathbb{E}[\Delta(\sigma(t))] = \mathbb{E}[L(\sigma(t)) - L(\sigma(t-1))] \leq \mathcal{Q}' - \delta \cdot \mathbb{E}[\sigma(t)]. \tag{30}$$

The time-average accumulated latency can be obtained by the sum of (30) of each time slot divided by the number of total time slots T:

$$\frac{1}{T}\lim_{T\to\infty}\sum_{t=0}^{T-1}\mathbb{E}[\sigma(t)] \leq \frac{\mathcal{Q}' - \frac{1}{T}\mathbb{E}[L(\sigma(T))]}{\delta} \leq \frac{\mathcal{Q}'}{\delta}. \tag{31}$$

Considering the fact that $O(\frac{\mathcal{Q}'}{\delta}) = O(\gamma)$, the time-average accumulated latency of CEDC-O algorithm is bounded by $O(\gamma)$. □

Based on Theorem 3, our CEDC-O algorithm finds the optimal solution to problem $\mathcal{P}_2$ when $\gamma \to \infty$. However, with the increase in γ, the accumulated latency increases. The CEDC-O algorithm then needs more time slots to converge the time-averaged system latency so that the constraint (13) can be fulfilled. The performance analysis of the CEDC-O algorithm is also experimentally validated in Section 5.

5 Simulation

We experimentally evaluate the performance of CEDC-O and the impacts of different parameters on its performance. All simulations were conducted on a Windows-10 machine.

TABLE 3
Parameter Settings

	n	$\|D\|$	MS	ω	$l_{\mathcal{T}}$	$\mathcal{L}$	γ
Set #1	8	4	5	0.25	2	2	1
Set #2	5,6,7,8,9,10	4	5	0.25	2	0.8	1
Set #3	8	2,3,4,5,6	5	0.25	2	0.8	1
Set #4	8	4	2,3,4,5,6	0.25	2	0.8	1
Set #5	8	4	5	0.15,0.20,0.25,0.30,0.35	2	0.8	1
Set #6	8	4	5	0.25	0,1,2,3,4	0.8	1
Set #7	8	4	5	0.25	2	0.4,0.8,1.2,1.6,2	1
Set #8	8	4	5	0.25	2	0.8	0.5,1.0,1.5

5.1 Settings

5.1.1 Data Set

The simulations are conducted based on the widely-used real-world EUA data set [2]. This data set contains the geographical locations of 125 cellular base stations and 816 mobile users around those base stations in the Melbourne central business district area. In all sets of simulations, a certain number of edge servers are randomly selected from the data set. In each time slot, the total number of app users' data requests is randomly generated following a normal distribution $X \sim \mathcal{N}(\mu', \sigma'^2)$, where μ' is $\frac{|M|}{2}$ and σ' is $\frac{|M|}{4}$, to cover 99.99 percent possibility. All the data requests are independently and identically distributed. The links between edge servers are randomly generated but we ensure that the edge servers constitute a connected graph.

To reflect the advantage of EC over cloud computing in terms of latency, the latency between the cloud and the app user's mobile devices in this graph area is 20 hops. This number is area-specific and does not impact the experimental results significantly as long as it is adequately large. To run the simulations realistically, we adopt AWS's Snowball Edge Pricing,[3] and set c_{mc} to $\$0.016$, c_{ms} to $\$0.008$ and c_s to $\$0.04$ caching per piece of data. The available storage space of each edge server is randomly generated separately following a normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, where μ is the half number of maximum cache spaces and σ is 1, to build the standard normal distribution covering all the possibilities.

5.1.2 Benchmark Approaches

We compare our new CEDC-O algorithm with four representative approaches: *Delay-Oriented*, *Online-Optimal*, *Revenue-Oriented* and *Coverage-Oriented*:

- *Delay-Oriented data caching approach (DO)*: this approach always finds the optimal solution to minimize the total data latency of all users. The data caching strategy is found by this approach. Since This approach originated from [28].
- *Online-Optimal data caching approach (OO)*: this approach finds the optimal solution to the CEDC problem based on $\mathcal{P}_1$ in each individual time slot. Since (13) is a long-term latency constraint, we use (32) as a constraint to drive OO in individual time slots:

$$\frac{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}\cdot l^t_{m,d}}{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}} \leq \mathcal{L}. \tag{32}$$

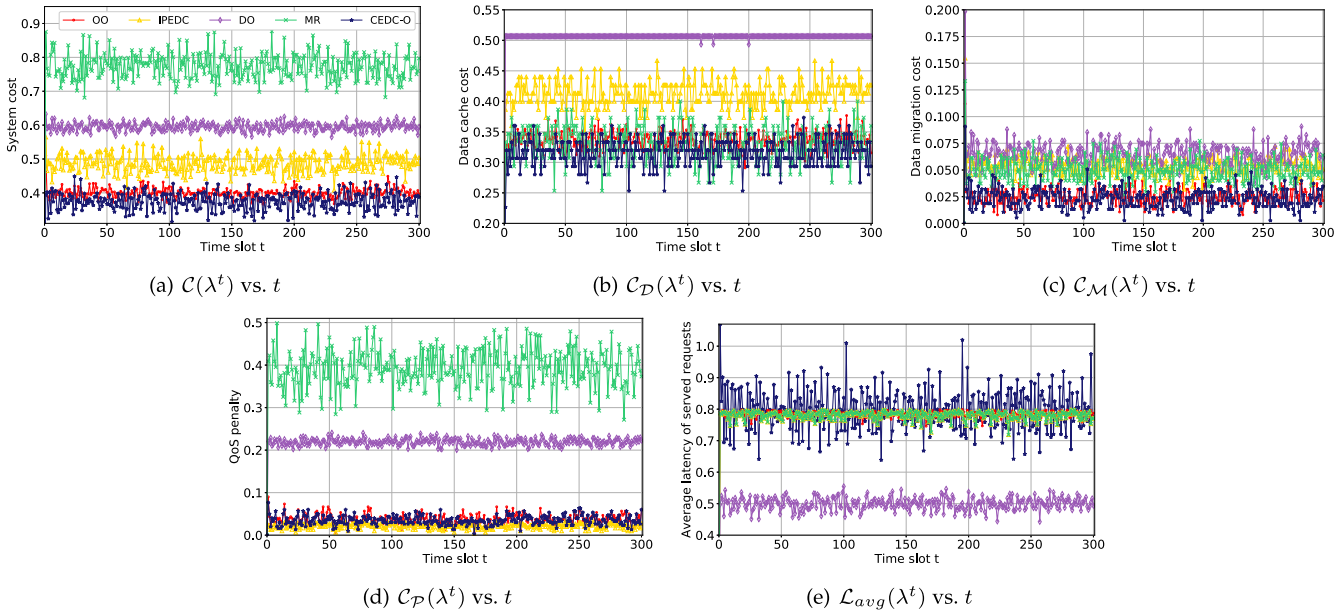
- *IPEDC* [33]: this approach minimizes the data caching cost to cover nearby users. Since some of the users may not be covered in our scenarios, we modify the original IPEDC approach to cover as many users as possible without violating the latency constraint (32).
- *Maximum Revenue data caching approach (MR)* [34]: This approach calculates the data caching revenue the benefits minus the costs produced by the data caching strategy in the EC environment. It always finds the optimal solution with the maximum data caching revenue. To perform a fair comparison in CEDC scenarios, latency constraint (32) is included into MR.

5.1.3 Parameter Settings

to analyze the performance of our CEDC-O comprehensively, we conducted seven sets of simulations to observe its performance in different CEDC scenarios. In each set of simulations except Set #1, we change one setting parameter and fix the other six. The simulation settings are summarized in Table 3. This way, we can compare the performance of the four approaches and observe how the changes in the setting parameters impact the performance of CEDC-O. The total number of time slots is 300 in all the simulations. Each time a setting parameter varies as follows, the simulation is repeated 20 times and the results are averaged:

- Number of edge servers (n). This parameter impacts the size of graph G and varies from 5 to 10 in steps of 1.
- Number of data ($|D|$). The total number of data to be cached over G, varies from 2 to 6 in steps of 1.
- Number of maximum cache spaces (MS). This parameter impacts the available cache spaces on edge server and varies from 2 to 6 in steps of 1.
- Ratio of c_p over c_s (ω) in (11). This parameter indicates the app vendor's priority for QoS and increases from 0.15, 0.20, 0.25, 0.30 to 0.35.
- Latency limit (l_T). This parameter varies from 0 to 4 in steps of 1. Specifically, $l_T = 0$ means that edge servers cannot communicate with each other - users can only access data from their local edge servers.

3. https://aws.amazon.com/snowball-edge/pricing/



(a) $\mathcal{C}(\lambda^t)$ vs. t (b) $\mathcal{C}_{\mathcal{D}}(\lambda^t)$ vs. t (c) $\mathcal{C}_{\mathcal{M}}(\lambda^t)$ vs. t

(d) $\mathcal{C}_{\mathcal{P}}(\lambda^t)$ vs. t (e) $\mathcal{L}_{avg}(\lambda^t)$ vs. t

Fig. 3. Simulation Set #1.

- Long-term latency ($\mathcal{L}$) in (15). This parameter varies from 0.4 to 2 in steps of 0.4.
- Trade-off parameter (γ) between $\mathcal{C}(\lambda^t)$ and $\sigma(t)$ in $\mathcal{P}_2$. As discussed in Section 4.2, this parameter impacts the performance bound of CEDC-O and increases from 0.5, 1.0 to 1.5.

5.1.4 Performance Metrics

In these simulations, five performance metrics are employed to evaluate all approaches:

- System cost ($\mathcal{C}(\lambda^t)$), the lower the better.
- Data caching cost ($\mathcal{C}_{\mathcal{D}}(\lambda^t)$), the lower the better.
- Data migration cost ($\mathcal{C}_{\mathcal{M}}(\lambda^t)$), the lower the better.
- QoS penalty ($\mathcal{C}_{\mathcal{P}}(\lambda^t)$), the lower the better.
- Number of served requests (Srn), the higher the better.

5.2 Performance Comparison

Fig. 3 presents the results of Set #1. Overall, of all the four approaches, *CEDC-O achieves the lowest system cost*. Fig. 3a depicts that, in term of the system cost in each time slot, the advantages of CEDC-O are 8.60 percent over OO, 23.05 percent over IPEDC, 37.04 percent over DO and 51.58 percent over MR. The advantage of CEDC-O over OO is not as significant because OO finds the optimal solution to the CEDC problem in each individual time slot. However, in 199 out of the 300 time slots in the experiments, CEDC-O achieves a system cost lower than than achieved by OO. This shows the overall advantage of CEDC-O over OO over time.

In Fig. 3b, the average data caching cost of CEDC-O is again the lowest at 0.3166, while the average data cache costs of OO, IPEDC, DO and MR are 0.3373, 0.4128, 0.5065 and 0.3314, respectively. Interestingly, the performance of DO in this figure is almost a horizontal line. DO always tries to achieve the lowest latency without considering the used cache space. Thus, it exhausts all the available cache spaces in most of the time slots.

Fig. 3c demonstrates the data migration costs of the five approaches over individual time slots. The migration costs of all the approaches are very high at the beginning, i.e., 0.0907 for CEDC-O, 0.1120 for OO, 0.1547 for IPEDC, 0.2204 for DO and 0.1333 for MR. This is because all the cached data are migrated from the cloud in time slot 0. After the data are cached on edge servers, the migration cost decreases because required data are already cached on edge servers.

Fig. 3d shows that IPEDC achieves the lowest QoS penalty, closely followed by CEDC-O and OO. The average QoS penalties of all the approaches are 0.035 for CEDC-O, 0.0373 for OO, 0.0205 for IPEDC, 0.2193 for DO and 0.6548 for MR. IPEDC focuses on covering the maximum number of users with available cache spaces. Thus, it achieves the lowest QoS penalty.

In terms of the average system latency of served requests over the edge server network, all the approaches fulfill constraint (13) as shown in Fig. 3e. The average latency is 0.7975 for CEDC-O, 0.7842 for OO, 0.7804 for IPEDC 0.4999 for DO and 0.7744 for MR. Fig. 3e also shows that the performances of OO, IPEDC and MR fluctuate slightly around 0.8. The reason is that the time-averaged latency achieved by these approaches are limited by (32).

5.3 Impact of Edge Server Number

Fig. 4 demonstrates the results of simulation Set #2, where the number of edge servers varies. Again, *CEDC-O outperforms the other four approaches* in terms of system cost per time slot, by 7.36 percent against OO, 23.14 percent against IPEDC, 46.43 percent against DO and 56.45 percent against MR. Since the number of users is determined by the number of edge servers selected from the EUA dataset, the number of users and the number of data requests increase accordingly when the number of edge servers increases. Thus, the

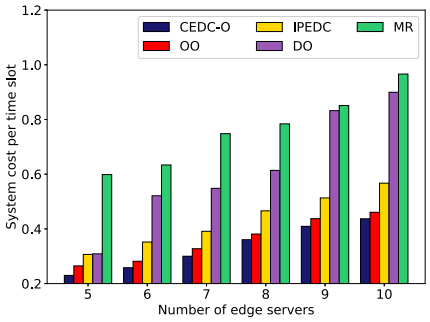


Fig. 4. Simulation Set #2.

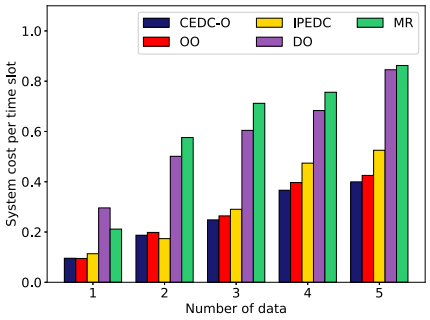


Fig. 5. Simulation Set #3.

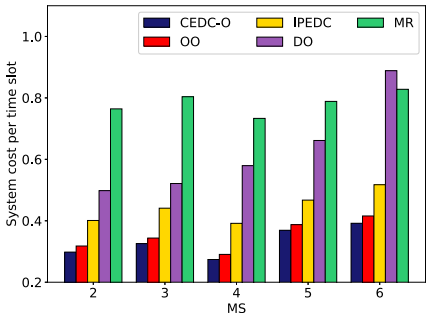


Fig. 6. Simulation Set #4.

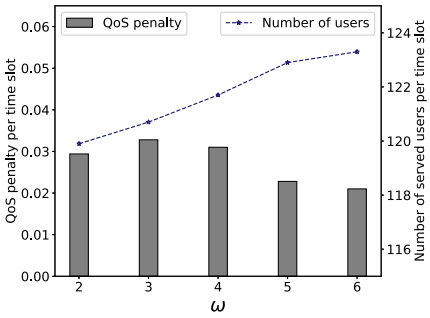


Fig. 7. Simulation Set #5.

system costs of all the approaches increases when the number of edge servers increases, as shown in Fig. 4.

5.4 Impact of Data Number

Fig. 5 depicts the results of simulation Set #3. When the number of data varies, *CEDC-O again achieves the lowest average system cost* per time slot. When the number of data increases from 1 to 5, the system costs per time slot achieved by all the five approaches increase, from 0.0956 to 0.3996 for CEDC-O, from 0.0948 to 0.4252 for OO, from 0.1136 to 0.5252 for IPEDC, from 0.2960 to 0.8456 for DO and from 0.2116 to 0.8624 for MR. With the increase in the number of data, app users are more likely to request different data from across multiple time slots. Accordingly, the average system costs achieved by all the five approaches increase.

5.5 Impact of Maximum Cache Spaces

In simulation Set #4, *CEDC-O achieves the lowest system cost per time slot* at *the lowest data cache cost per time slot*. The advantages of CEDC-O in the system cost per time slot are 5.50 percent over OO, 25.23 percent over IPEDC, 47.32 percent over DO and 57.66 percent over MR. The costs spent on data caching per time slot increase from 0.29 to 0.42 for CEDC-O, from 0.40 to 0.85 for DO and from 0.30 to 0.48 for OG, when the number of maximum cache spaces increases from 2 to 6. Different from other approaches, the system cost per time slot of DO always increases when the number of maximum cache spaces increases from 2 to 6. The reason is that DO focuses on latency optimization instead of cost optimization, and thus always exhausts the available cache spaces.

5.6 Impact of QoS Priority

Fig. 7 shows the impact of QoS priority on the performance of CEDC-O in terms of the QoS penalty and the number of served users over the edge server network. The average number of users served over the edge server network per time slot increases from 119.9 to 123.3, where ω increases from 2 to 6. The reason is that the more data cache spaces are hired to serve more users to reduce the QoS penalty. However, the QoS penalty increases from 0.0294 to 0.0328 when ω increases from 2 to 3, then decreases to 0.0210 when ω increases from 3 to 6. This is because the increase in the QoS penalty caused by the increasing ω is more than the reduction caused by hiring more cache spaces when pursuing the objective of the minimum system cost.

5.7 Impact of Latency Limit

Fig. 8 shows the QoS penalty and the number of served users over the edge server network per time slot when the latency limit $l_{\mathcal{T}}$ varies. When $l_{\mathcal{T}}$ increases from 0 to 4, the QoS penalty rapidly decreases from 0.6810 to 0, while the average number of served users increases from 72.8 to 130.9. This is because most of the users can access more edge servers when the latency limit increases. Specifically, there are 8 edge servers in Set #6, and many users can access all those edge servers within 3 hops. Thus, all the requests can be fulfilled and the QoS penalty is 0.

5.8 Impact of long-Term Latency

Fig. 9 illustrates the results of Set #7 when the long-term latency constraint varies. The system cost per time slot of CEDC-O decreases from 0.5848 to 0.2772 when $\mathcal{L}$ increases from 0.4 to 1.6. It stabilizes when $\mathcal{L}$ increases from 1,6 to 2.0. The main reason is that the same data replica can be transmitted to more users when $\mathcal{L}$ increases. In terms of the average latency of served requests $\mathcal{L}_{avg}(\lambda^t)$, it stabilizes for the same reason when $\mathcal{L}$ increases from 1,6 to 2.0.

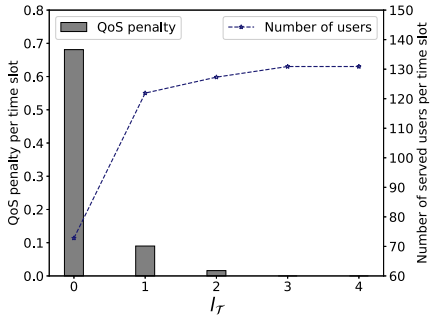


Fig. 8. Simulation Set #6.

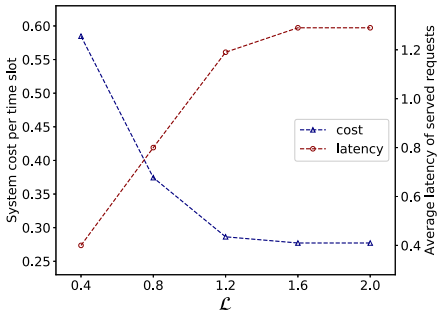


Fig. 9. Simulation Set #7.

5.9 Impact of Trade-Off Parameter

Fig. 10 shows the impact of the trade-off parameter γ in $\mathcal{P}_2$ on the time-averaged latency, which is the left side of (13). As discussed in Section 4.2, CEDC-O needs more time slots to satisfy constraint (13) when γ increases. With $\gamma = 0.5$, $\gamma = 1.0$ and $\gamma = 1.5$, the time-averaged latency achieved by CEDC-O converges back to $\mathcal{L} = 0.8$ in time slots 4, 100 and 195, respectively. In the small-scale figure, the blue line overtakes the red one in the 5th time slot. This is because of $\Delta(\sigma(t)) = 0$ in the 4th time slot and that no penalty is produced in the next time slot. In this case, the objective becomes $\min(\mathcal{Q} + \gamma \cdot \mathcal{C}(\lambda^t))$ and the time-averaged latency increases significantly in the 5th time slot. In the meantime, CEDC-O with $\gamma = 1.0$ still tries to converge the time-averaged latency back to $\mathcal{L}$. After the 5th time slot, the blue line is always lower than the red one, while the red one is lower than the yellow one. In conclusion, the system stability, ensured by the long-term latency constraint (13) (same as the accumulated latency requirement (16)), is ensured by the CEDC-O algorithm with different trade-off parameters.

5.10 Threats to Validity

5.10.1 Construct Validity

The major threat to construct validity is the four approaches used for comparison. Due to the novelty of the CEDC problem in the EC environment, DO has a different objective from CEDC and OO only considers the current time slot, while IPEDC and MR do not consider the long-term latency constraint. Thus, there is a threat that the comparison does not suffice to comprehensively evaluate CEDC-O. To minimize this threat, we enhanced IPEDC and MR by including (32) into their implementation. Moreover, we changed seven parameters, as presented in Table 3, to simulate various CEDC scenarios. In this way, we could evaluate our approaches by not only the comparison to the other approaches, but also the demonstration of how the changes in the parameters impact the performance of CEDC-O.

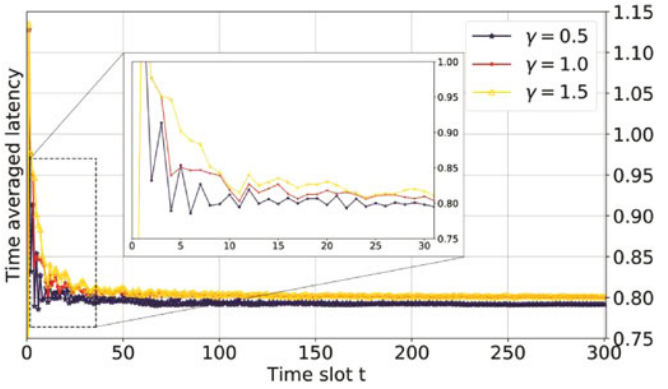


Fig. 10. Simulation Set #8.

5.10.2 External Validity

The main threat to the external validity of the evaluation is whether our approaches can be generalized and applied in other CEDC scenarios in the EC environment. To tackle this threat, We measured the performance of our approaches in a generic way. Specifically, we measured the data sizes and cache spaces by the number of data units and the data retrieval latency by the number of hops. In this way, the evaluation results can be interpreted with specific models for all cost components. In addition, we ran the simulations on a widely-used real-world data set while varying seven parameters to vary the size and the complexity of the CEDC problem. This way, the representativeness and comprehensiveness of the evaluation are ensured.

6 Related Work

Data caching have been extensively investigated in the fields of conventional distributed computing and cloud computing environments. With the popularity of edge computing, data caching in the edge computing environment is obtaining attention from researchers recently.

6.1 Conventional Distributed Data Caching

In the last few decades, there are many data caching problems investigated in conventional distributed computing environments, including web caching [13], content delivery network [35], etc. Banerjee *et al.* [36] developed a content placement strategy for information-centric network based on data popularity, namely Greedy Caching. With popular contents cached in the network, the Greedy Caching approach considered the cache miss rate at the edge to decide what contents would be cached on the core server. In [37], the authors formulated two caching strategies for data publish-subscribe systems, including eviction-based caching and time-to-live-based caching to address the space and time issues, respectively. The authors of [38] focused on balancing the trade-off between latency and cost in the content-centric network. They addressed this issue with a holistic model for provisioning the storage capability based on the network performance and the provisioning cost.

6.2 Cloud Data Caching

In the cloud computing environment, a critical problem of data caching is how to utilize cache space efficiently on cloud hosts and mobile devices. Arteaga *et al.* [39] proposed CloudCache, a method for managing cache, to fulfill the caching requirement of the workload and minimize cache wear-out. In [40], the authors presented how to use segment access-aware dynamic semantic cache in the cloud computing environment for relational databases. A cache access algorithm was introduced to consider cache exact hit, cache extended hit, cache partial hit and cache miss. The authors of [41] explored the cache design space for embedded processors with evolutionary techniques for mobile and thin client processors in the cloud computing environment. A heuristic and evolutionary method was presented to generate a near-optimal cache space design for enhancing service quality. In [42], the authors formulated a benefit maximization problem and created a cache replacement approach based on traffic requirements. They also introduced a content clustering method for collecting popular data and clustering similar contents.

6.3 Edge Data Caching

Edge computing (EC) extends cloud computing with computing resources and services geographically distributed at the edge of the cloud [43]. With the deployment of edge servers, the problem of computation offloading arises. It has been well studied with consideration of edge servers' energy efficiency, offloading cost and joint caching [28], [44].

Recently, the challenges raised by data caching are being investigated in the EC environment. Existing data caching approaches are rendered obsolete by the new characteristics of EC and thus cannot be directly applied in the EC environment. Thus, researchers are proposing and investigating new ideas and techniques for data caching in the EC environment. Cao *et al.* present an optimal auction mechanism to maximize the service provider's revenue based on cache allocation and user valuation reports. They propose computationally efficient approaches to apply the auction mechanism based on data retrieval and delivery costs. The authors of [45] propose a caching system named Cachier for recognition applications in the EC environment. Cachier coordinates the loading balance between edge servers and the cloud to minimize the data retrieval latency dynamically. However, the above approaches employ offline methods and require complete information about active users and data requests in all time slots. They cannot handle edge data caching scenarios where data and users may come and go randomly.

Instead of solving the edge data caching problem optimally in an offline manner, some researchers investigate online approaches for solving the dynamic edge data caching problems. Xu *et al.* [28] propose an online algorithm named OREO to decide service caching and task offloading. The system aims to minimize the total network latency and applies a long-term energy consumption constraint to stabilize the edge caching system. The authors of [46] propose MOREA, an online algorithm considering user mobility, to allocate different resources like caches and CPUs on edge servers for computation offloading. In [47], the authors integrate the cloud radio access network with the EC technology to schedule resources including caches and computational resources dynamically. They propose the VariedLen algorithm to maximize the mobile network provider's profit. They also extend the standard Lyapunov technology so that individual tasks can be performed across multiple time slots. However, existing works investigate edge data caching only to complement computing offloading and fail to give data caching sufficient attention as a unique technology with advantages in reducing data retrieval latency and improving the quality of services and users' experiences, especially from the app vendor's perspective who is an important stakeholder in the EC environment.

Edge computing inherits the pay-as-you-go pricing model from cloud computing, which allows app vendors to hire storage resources on edge servers from edge infrastructure providers to cache app data for their users. Thus, the cost incurred by data caching for app vendors is critical to the success of edge computing because, after all, app vendors are the main customers in the edge computing environment. To the best of our knowledge, this paper makes the first attempt to propose an approach named CEDC-O for solving the CEDC problem from the app vendor' perspective in the EC environment. By innovatively and realistically modeling the CEDC problem as a long-term optimization problem, CEDC-O can help app vendors ensure the long-term performance of their edge data caching performance.

7 Conclusion

In this paper, we studied the collaborative edge data caching (CEDC) problem. We first identified the major challenges and proposed a comprehensive cost model for this problem, where system cost is composed of data caching cost, data migration cost and QoS penalty. We also proved the $\mathcal{NP}$-completeness of the CEDC problem. We proposed CEDC-O, an online algorithm with provable performance guarantee, and evaluated its performance with extensive simulations. This research has established the foundation for the CEDC problem and opened up a number of future research directions. In our future work, we will consider dynamics on available edge server caches, user mobility and security policies.

Acknowledgments

This work was funded in part by Australian Research Council Discovery Projects (DP180100212 and DP200102491) and Laureate Fellowship FL190100035.

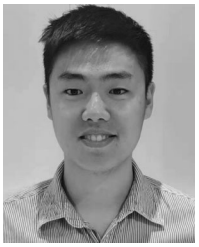
References

[1] A. Osseiran *et al.*, "The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions," in *Proc. IEEE 77th Veh. Technol. Conf.*, 2013, pp. 1–5.

[2] P. Lai *et al.*, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2018, pp. 230–245.

[3] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, vol. 12, no. 4, pp. 29–33, Jul. 2017.

[4] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[5] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.
[6] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[7] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[8] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.
[9] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," *Computer*, vol. 23, no. 6, pp. 12–24, 1990.
[10] J. D. Owens *et al.*, "A survey of general-purpose computation on graphics hardware," in *Computer Graphics Forum*, vol. 26. Hoboken, NJ, USA: Wiley, 2007, pp. 80–113.
[11] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. San Mateo, CA, USA: Morgan Kaufmann, 2010.
[12] A. J. Smith, "Disk cachemiss ratio analysis and design considerations," *ACM Trans. Comput. Syst.*, vol. 3, no. 3, pp. 161–203, 1985.
[13] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.
[14] K. Elhardt and R. Bayer, "A database cache for high performance and fast restart in database systems," *ACM Trans. Database Syst.*, vol. 9, no. 4, pp. 503–525, 1984.
[15] A. Mukhopadhyay, N. Hegde, and M. Lelarge, "Optimal content replication and request matching in large caching systems," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 288–296.
[16] K. Ji, G. Quan, and J. Tan, "Asymptotic miss ratio of LRU caching with consistent hashing," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 450–458.
[17] G. Casale, "Analyzing replacement policies in list-based caches with non-uniform access costs," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 432–440.
[18] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2012, pp. 316–321.
[19] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1029–1040, Aug. 2015.
[20] M. Dehghan *et al.*, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1635–1648, Jun. 2017.
[21] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.
[22] C.-Y. Wang, S. H. Lim, and M. Gastpar, "Information-theoretic caching: Sequential coding for computing," *IEEE Trans. Inf. Theory*, vol. 62, no. 11, pp. 6393–6406, Nov. 2016.
[23] M. Patel *et al.*, "Mobile edge computing-introductory technical white paper," *Mobile-Edge Comput. (MEC) Industry Initiative*, White Paper, pp. 1089–7801, 2014.
[24] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018.
[25] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[26] M. Chen, Y. Hao, K. Lin, Z. Yuan, and L. Hu, "Label-less learning for traffic control in an edge network," *IEEE Netw.*, vol. 32, no. 6, pp. 8–14, Nov./Dec. 2018.
[27] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.
[28] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.
[29] T. Zhang, F. Ren, and R. Shu, "Towards stable flow scheduling in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2627–2640, Nov. 2018.
[30] N. Abedini and S. Shakkottai, "Content caching and scheduling in wireless networks with elastic and inelastic traffic," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 864–874, Jun. 2014.
[31] E. Hazan, S. Safra, and O. Schwartz, "On the complexity of approximating k-set packing," *Comput. Complexity*, vol. 15, no. 1, pp. 20–39, 2006.
[32] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–211, 2010.
[33] X. Xia *et al.*, "Graph-based optimal data caching in edge computing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2019, pp. 477–493.
[34] Y. Liu, Q. He, D. Zheng, M. Zhang, F. Chen, and B. Zhang, "Data caching optimization in the edge computing environment," in *Proc. IEEE Int. Conf. Web Serv.*, 2019, pp. 99–106.
[35] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter , "AdaptSize: Orchestrating the hot object memory cache in a content delivery network," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 483–498.
[36] B. Banerjee, A. Kulkarni, and A. Seetharam, "Greedy caching: An optimized content placement strategy for information-centric networks," *Comput. Netw.*, vol. 140, pp. 78–91, 2018.
[37] M. Y. S. Uddin and N. Venkatasubramanian, "Edge caching for enriched notifications delivery in big active data," in *Proc. 38th IEEE Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 696–705.
[38] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," in *Proc. 33rd IEEE Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 62–72.
[39] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "CloudCache: On-demand flash cache management for cloud computing," in *Proc. 14th USENIX Conf. File Storage Technol.*, 2016, pp. 355–369.
[40] K. Ma, B. Yang, Z. Yang, and Z. Yu, "Segment access-aware dynamic semantic cache in cloud computing environment," *J. Parallel Distrib. Comput.*, vol. 110, pp. 42–51, 2017.
[41] A.-H. A. Badawy, G. Yessin, V. Narayana, D. Mayhew, and T. El-Ghazawi, "Optimizing thin client caches for mobile cloud computing: Design space exploration using genetic algorithms," *Concurrency Comput.: Practice Experience*, vol. 29, no. 11, 2017, Art. no. e4048.
[42] S. Tamoor-ul Hassan , S. Samarakoon, M. Bennis, M. Latva-Aho , and C. S. Hong, "Learning-based caching in cloud-aided wireless networks," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 137–140, Jan. 2018.
[43] M. Yannuzzi *et al.*, "A new era for cities with fog computing," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 54–67, Mar./Apr. 2017.
[44] S. Jošilo and G. Dán, "A game theoretic analysis of selfish mobile computation offloading," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[45] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 276–286.
[46] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "MOERA: Mobility-agnostic online resource allocation for edge computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1843–1856, Aug. 2019.
[47] X. Wang *et al.*, "Dynamic resource scheduling in mobile edge cloud with cloud radio access network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2429–2445, Nov. 2018.



Xiaoyu Xia received the master's degree from the University of Melbourne, Australia, in 2015. He is currently working toward the PhD degree at Deakin University. His research interests include edge computing, service computing, and software engineering.

Feifei Chen received the PhD degree from the Swinburne University of Technology, Australia, in 2015. She is a lecturer with Deakin University. Her research interests include software engineering, cloud computing, and green computing. For more details please visit https://sites.google.com/view/feifeichen/.

Qiang He (Senior Member, IEEE) received the first PhD degree from the Swinburne University of Technology, Australia, in 2009, and the second PhD degree in computer science and engineering from the Huazhong University of Science and Technology, China, in 2010. He is a senior lecturer with Swinburne. His research interests include service computing, software engineering, cloud computing, and edge computing. For more details please visit https://sites.google.com/site/heqiang/.

John C. Grundy (Senior Member, IEEE) received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently an Australian Laureate fellow and a professor of software engineering with Monash University, Melbourne, Australia. He is an associate editor of the *IEEE Transactions on Software Engineering*, *Automated Software Engineering Journal*, and *IEEE Software*. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. For more details please visit https://sites.google.com/site/johncgrundy/.

Mohamed Abdelrazek received the PhD degree from Swinburne University, in 2014. He is currently an associate professor with software engineering and IoT with the School of Information Technology, Deakin University, Australia. He has more than 10 years experience in building software solutions. His research interests include software engineering, security, and artificial intelligence. For more details please visit https://sites.google.com/site/mohamedalmorsy/.

Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology, in 1994. He is a Cheung Kung Scholars chair professor of computer science and engineering with the Huazhong University of Science and Technology (HUST), in China. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

4.2 Effective, Efficient and Cost-effective Data Replacement Strategies in Edge Computing

Except for the cost, the benefit produced by placing and replacing data is also important to the success of app vendors. The key challenge for app vendors is to formulate more cost-effective strategies, considering both the caching benefit obtained from providing low-latency data retrieval to its users and the caching cost incurred based on the pay-as-you-go scheme. In this chapter, we formally formulate this online edge data replacement problem and prove its NP-hardness. To solve this problem effectively and efficiently, we propose an online algorithm named OL-MEDC, without requiring future information. We compare OL-MEDC with five representative approaches, including the Lyapunov-based on presented in **Chapter 4.1**. The experimental results show OL-MEDC's advantages over these approaches in addressing this online edge data replacement problem.

This chapter is based on a published paper, entitled: OL-MEDC: An Online Approach for Cost-effective Data Caching in Mobile Edge Computing Systems, *IEEE Transactions on Mobile Computing*, accepted in 2021.

OL-MEDC: An Online Approach for Cost-effective Data Caching in Mobile Edge Computing Systems

Xiaoyu Xia, Feifei Chen, Qiang He, *Senior Member, IEEE*, Guangming Cui, John Grundy, *Senior Member, IEEE*, Mohamed Abdelrazek, Athman Bouguettaya, *Fellow, IEEE*, and Hai Jin, *Fellow, IEEE*

Abstract—Mobile Edge Computing (MEC) has emerged to overcome the inability of cloud computing to offer low latency services. It allows popular data to be cached on edge servers deployed within users' geographic proximity. However, the storage resources on edge servers are constrained due to their limited physical sizes. Existing studies of edge caching have predominantly focused on maximizing caching performance from the mobile network operator's perspective, e.g., maximizing data retrieval success rate, minimizing system energy consumption, balancing the overall caching workload, etc. App vendors, as key stakeholders in MEC systems, need to maximize the caching revenue, considering the cost incurred and the benefit produced. We investigate this novel Mobile Edge Data Caching (MEDC) problem from the app vendor's perspective, and prove its $\mathcal{NP}$-hardness. We then propose Online MEDC (OL-MEDC), an approach that formulates MEDC strategies for app vendors, without requiring future information about data demands. Its performance is theoretically analyzed and experimentally evaluated. The experimental results demonstrate that OL-MEDC outperforms state-of-the-art approaches by at least 20.41% on average.

Index Terms—data caching, mobile edge computing, online algorithm, cost-effective.

1 Introduction

The world is witnessing an exponential growth of mobile and Internet-of-Things devices in recent years [1], [2]. These devices generate enormous network traffic, often increase network latency and cause network congestion. To tackle this challenge, Mobile Edge Computing (MEC) has emerged to distribute computing and storage resources at the network edge by deploying edge servers at base-stations within end-users' geographic proximity [3]. Mobile and Internet-of-Things application vendors – referred to as *app vendors* hereafter – can request the use of computing and storage resources on edge servers to deploy applications for ensuring low-latency service and data accesses for their users [4].

As the number of end-users accessing edge applications increases, it is expected that a large amount of app data will be transmitted via edge servers between remote cloud servers and users' end-devices. Caching app vendors' data on edge servers, (e.g. viral videos), will significantly reduce their users' data retrieval latency because the users can retrieve data directly from edge servers within their close geographic proximity.

Data caching techniques have been intensively studied and applied to leverage their advantages in saving bandwidth consumption, minimizing access costs and reducing network latency [5], [6], [7]. In the last few years, there has been intensive research investigating network cache in the conventional network paradigms relying on different perspectives, e.g., information-theoretic caching [8] and request routing [9]. As a new computing paradigm, MEC offers new opportunities but poses new challenges for data caching. In MEC systems, the fundamental goal is to lower users' data retrieve latency by caching popular data on edge servers [10].

Research has started in earnest to study data caching problems in MEC systems – referred to as the Mobile Edge Data Caching (MEDC) problem hereafter – from the mobile network operator's perspective with different offline optimization objectives, e.g., minimum response latency [11], or maximum data sharing efficiency [12]. However, these studies have not systematically considered the requirements and concerns of app vendors like Facebook or Uber who posses storage resources on edge servers to cache data. In the MEC environment, app vendors and mobile network operators are two stakeholders with quite different interests. From the mobile network operator's perspective, the main objective is to optimize the overall caching performance in edge data caching scenarios, e.g., maximizing data retrieval success rate [13], minimizing system energy consumption [14], or balancing the overall caching workload [15]. In addition to mobile users, app vendors are the mobile network operators' other group of customers in the MEC environment. The key challenge for app vendors is to formulate

- *X. Xia, F. Chen and M. Abdelrazek are with School of Information Technology, Deakin University, Australia. E-mail: xiaoyu.xia@deakin.edu.au; feifei.chen@deakin.edu.au; mohamed.abdelrazek@deakin.edu.au.*
- *Q. He and G. Cui are with School of Software and Electrical Engineering, Swinburne University of Technology, Australia. E-mail: qhe@swin.edu.au; gcui@swin.edu.au.*
- *J. Grundy is with Faculty of Information Technology, Monash University, Australia. E-mail: john.grundy@monash.edu.*
- *A. Bouguettaya is with School of Computer Science, University of Sydney, Australia. Email: athman.bouguettaya@sydney.edu.au.*
- *H. Jin is with School of Computer Science and Technology, Huazhong University of Science and Technology, China. Email: hjin@hust.edu.cn.*

more cost-effective caching strategies, considering both the caching benefit obtained from providing low-latency data retrieval to its users and the caching cost incurred based on the pay-as-you-go scheme.

Given a set of requested data in an area, caching all the requested data on every individual edge server is a straightforward solution for app vendors to ensure low latency for all the users. However, edge servers' cache capacities are normally limited and expensive because of their physical size limits [16]. It is difficult to cache a large amount of data for each app vendor on each edge server because of the competitions among app vendors. A common practice is to reserve cache capacities on individual edge servers in advance. Users may then benefit from low-latency data retrievals provided by their nearby edge servers, producing *caching benefits* for the app vendor. In the meantime, *caching costs* are incurred based on the pay-as-you-go scheme when data are transmitted to be cached [17]. Both caching benefits and caching costs must be considered when cost-effectively formulating MEDC strategies for app vendors. In addition, different data will need to be cached dynamically over time. Cached data may need to be flushed out to save cache spaces for more popular data. These dynamic data demands are not known in advance and must be accommodated on the fly. Thus, offline MEDC approaches are subject to low effectiveness over time.

In this paper, *we study this online MEDC problem from the app vendor's perspective for maximizing the caching revenue, considering both caching benefits and caching costs.* The main contributions include:

- We formulate this online MEDC problem and prove its $\mathcal{NP}$-hardness.
- We propose OL-MEDC (Online MEDC), an online approach for formulating cost-effective MEDC strategies over time, to solve the MEDC problem over time without requiring future information about data demands.
- We analyze the guarantee bound of OL-MEDC theoretically.
- We compare OL-MEDC with five representative approaches through extensive experiments, and show OL-MEDC's advantages over these approaches in addressing the online MEDC problem.

The organization of the paper is as follows. An example motivating the MEDC problem is provided in Section 2. In Section 3, we present the system model, formulate the MEDC problem and prove that the MEDC problem is $\mathcal{NP}$-hard. In Section 4, we present the design of OL-MEDC, and prove its performance guarantee theoretically. Section 5 evaluates OL-MEDC experimentally against five representative approaches, analyzes the construct, internal and external threats to the validity of the evaluation, and discuss how they are mitigated. The related work is reviewed in Section 6. We conclude this study and point out the future work in Section 7.

2 Motivating Example

In an MEC environment, edge servers in an area can transmit data with other via high-speed links [4]. Those edge servers and links constitute an edge server network [17].

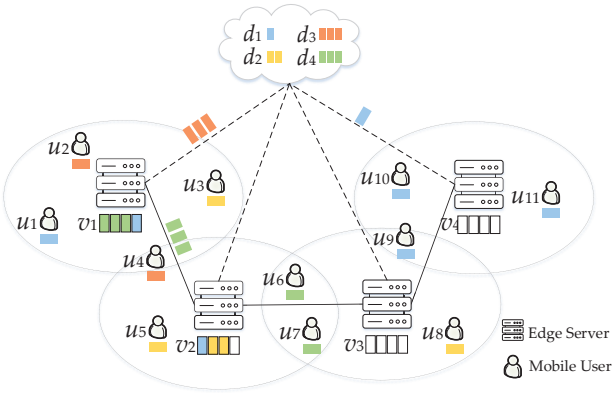


Fig. 1: An example MEDC scenario

This architecture single-point failure problem in the edge-cloud architecture, where a centralized macro base station is used to control the communications between edge servers [18]. It also avoids unpredictable network latency when edge servers can only communicate the backhaul network [17]. In fact, the edge-cloud architecture is a fully-connected edge server network. Thus, OL-MEDC can also accommodate the edge-cloud architecture.

Fig. 1 shows a typical MEC system, involving 4 edge servers, $\{v_1, ..., v_4\}$ and 11 users, $\{u_1, ..., u_{11}\}$ that request 4 data, $\{d_1, ..., d_4\}$. Compared to cloud servers, the storage resources at the network edge are constrained by edge servers' limited physical capacities. As discussed in Section 1, caching plenty of data on each edge server is usually impossible due to edge servers' constrained storage resources and the consequent competition among app vendors [3]. This is the *capacity constraint*. Due to this constraint, an app vendor will not always be able to acquire the required cache spaces. Thus, app vendors normally need to reserve a certain number of cache spaces on every edge server in advance. Take Fig. 1 for example. The app vendor has reserved four cache spaces on edge server v_1. However, d_1, d_2 and d_3 are requested in the coverage area of v_1, and the cache spaces of v_1 do not suffices to cache those three kinds of data. Usually, reserving a large amount of cache spaces on an individual edge server is cost-ineffective because users' data demands for popular data vary from area to area [19]. The ability for edge servers to enable collaborative caching further reduces the app vendor's need to reserve large amounts of cache capacities on individual edge servers.

Each edge server in the MEC system has its coverage area, and only the users in this coverage area can access the edge server directly [20]. In an overlapping area covered by multiple edge servers, a user can access its local edge servers, i.e. the edge servers covering this user. For example, user u_4 in Fig. 1 can access both v_1 and v_2 directly. This is the *coverage constraint*. If a user cannot retrieve data from its local edge servers, this user can obtain the data from remote edge servers as long as the app-specific latency constraint is not violated [3], [21]. This is the *latency constraint*. Take u_6 in Fig. 1 for example. It can obtain data directly from its local edge servers v_2 and v_3, or remote edge servers v_1 and

v_4. If none of them has the requested data, the user has to retrieve it from the cloud. The main difference between these retrieval methods lies in the corresponding data retrieval latencies.

In addition to the above three unique constraints in MEC systems, the main difference between this study and existing studies is that this paper investigates the MEDC problem for app vendors, considering data dynamics in MEC systems. Please note that the cost of hiring cache spaces is fixed because cache spaces are reserved in advance by the app vendor. *The optimal MEDC strategy for app vendors should maximize the app vendor's caching revenue, i.e., caching benefit minus caching cost.* Fig. 1 shows that users u_9, u_{10} and u_{11} request data d_1. Caching d_1 on v_4 in its reserved cache spaces can reduce the data retrieval latency of those users, compared with retrieving data from the remote cloud. This produces the *caching benefit*. However, transmitting d_1 to v_4 is charged by the mobile network operator. This incurs the *caching cost*. Furthermore, the status of a dynamic real-world MEC system changes over time. For example, users may leave the system and new popular data may be requested by new users. The MEDC strategy must be updated accordingly to remain optimal, taking into account these system dynamics, the information of which is not available prior to their occurrences. Finding and implementing the global optimal MEDC strategy over a long period of time is impractical. The app vendor's MEDC must be updated over time in an online manner without the need for knowing future data dynamics in the real-world MEC system.

TABLE 1: Summary of Notations

Notation	Description
$\mathcal{A}_i$	reserved cache spaces on v_i
c	remote cloud server
c_{ce}	unit data transmission cost from cloud to edge server
c_{ee}	unit data transmission cost between edge servers
D	set of requested data
d_f	data f
E	set of edges in G
G	edge server network
$q^t_{m,f}$	binary variable indicating u_m requests for d_f in t
$\mathcal{S}$	data caching strategy over T
s^t	data caching strategy in t
$s^t_{i,f}$	binary variable indicating d_f will be cached on v_i at the end of t
t	time slot t
T	set of time slots
V	set of edge servers
v_i	edge server i
U	set of end-users' devices
U_j	set of end-users' devices covered by v_j
u_m	end-users' device m
γ	unit of caching benefit
$\Phi_{j,i}$	hops between edge server v_j and v_i
Φ_L	latency limit
$\Phi^t_{i,f}$	lowest latency transmitting d_f for v_i in t
$\Phi^t_{m,f}$	lowest latency of u_m retrieving d_f in t

3 System Model and Problem Formulation

In this section, we first introduce the system architecture, then define the data retrieval latency, caching cost, and caching benefit under constrains discussed in Section 2.

3.1 System Architecture

Let V denote the set of edge servers and E denote the set of links in an MEC system, the edge server network can be modeled as a graph $G(V, E)$. The notations are summarized in Table 1.

Let D denote the data requested in the MEC system in a specific time slot t, let $s^t = \{s^t_1, ..., s^t_n\}$ denote the MEDC strategy for time slot t, where $s^t_i = \{s^t_{i,f}, \forall d_f \in D\}$, and $s^t_{i,f}$ indicates whether d_f is decided to be cached on v_i in time slot t:

$$s^t_{i,f} = \begin{cases} 0 & \text{if } d_f \text{ is not cached on } v_i \text{ in } t \\ 1 & \text{if } d_f \text{ is cached on } v_i \text{ in } t \end{cases} \quad (1)$$

Let $q^t_{m,d}$ denote whether user u_m requests data d_f in time slot t:

$$q^t_{m,f} = \begin{cases} 0 & \text{if } u_m \text{ does not request } d_f \text{ in } t \\ 1 & \text{if } u_m \text{ requests } d_f \text{ in } t \end{cases} \quad (2)$$

The data cached on v_i must not exceed the reserved spaces on v_i in any time slot:

$$\sum_{d_f \in D} s^t_{i,f} \cdot |d_f| \leq \mathcal{A}_i, \forall v_i \in V, t \in T \quad (3)$$

where $|d_f|$ is the size of d_f. This is the *cache space constraint*.

3.2 Data Retrieval Latency

There are two main components in the data retrieval latency of a user, including the latency transmitting data between edge servers and the latency transmitting data to the user from an edge server covering this user. Since the latter is extremely low in 5G and is not influenced by the MEDC strategy, we do not consider this component in the model. Modeling the connected edge servers in the MEC system as a graph in the same way as [17], the number of hops in MEC systems is used to generically quantify data retrieval latency. The latency of user u_m retrieving data d_f in time slot t, denoted by $\Phi^t_{m,f}$, is calculated as follows:

$$\Phi^t_{m,f} = \min\{\Phi_{j,i}, s^t_{j,f} = 1, \forall v_j \in V \bigcup c\}, \forall u_m \in U_i \quad (4)$$

where v_j is the edge server caching d_f, v_i is the edge server covering u_m, and $\Phi_{j,i}$ is the minimum number of hops between v_j and v_i.

Remark: Similar to [3], [17], specific latency models can be easily integrated into OL-MEDC to accommodate various real-world MEDC scenarios. Specifically, $\Phi_{j,i}$ in (4) can be replaced with a latency function $\Phi(j, i, f)$ that represents the latency in delivering data d_f from v_j to v_i:

$$\Phi^t_{m,f} = \min\{\Phi(j, i, f), s^t_{j,f} = 1, \forall v_j \in V \bigcup c\}, \forall u_m \in U_i \quad (5)$$

Let Φ_L denote the app-specific latency constraint, i.e., the maximum data retrieval latency allowed. If $\Phi^t_{m,f}$ is higher than Φ_L, u_m will access d_f from the remote cloud.

3.3 Caching Cost

To cache a data d_f on an edge server in the MEC system, d_f can be transmitted from another edge server or from the cloud. Either way, the transmission cost occurs. The caching cost incurred by implementing an MEDC strategy s^t is the total transmission cost incurred by transmitting D according to s^t. As discussed in Section 2, the app vendor reserves cache spaces on edge servers in advance. Thus, the cost incurred by hiring cache spaces is fixed and thus does not need to be considered here.

Similar to (4), let $\Phi^t_{i,f} = min\{\Phi_{i,j}, s^{t-1}_{j,f} = 1, \forall v_j \in V\}$ denote the lowest latency in transmitting d_f to v_i in time slot t. There are two types of data transmissions: from the remote cloud server to an edge server and from an edge server to another. Let c_{ce} denote the unit cost of transmitting data from cloud to edge servers, and c_{ee} denote the unit cost of transmitting data between edge servers. If the cost incurred by transmitting d_f from the remote cloud server ($c_{ee} \cdot \Phi^t_{i,f} > c_{ce}$) is lower than that from another edge server in the MEC system ($c_{ce} < c_{ee} \cdot \Phi^t_{i,f}$), the data will be transmitted from the remote cloud to v_i. Thus, the cost of transmitting data d_f in time slot t, denoted as $\mathcal{C}(s^t)$, is:

$$\mathcal{C}(s^t) = \sum_{v_i \in V} \sum_{d_f \in D} |d_f| \cdot s^t_{i,f}(1 - s^{t-1}_{i,f}) \cdot cost^t_{i,f} \tag{6}$$

where $cost^t_{i,f} = \min\{c_{ee} \cdot \Phi^t_{i,f}, c_{ce}\}$ is the minimum cost for delivering d_f to v_i in time slot t.

Remark: Similar to latency, specific transmission cost models can be easily integrated into OL-MEDC. A cost function $c(j, i, f)$ that represents the cost of delivering d_f from v_j to v_i can be employed to calculate the actual minimum cost of delivering d_f to v_i in time slot t:

$$cost^t_{i,f} = \min\{c(j, i, f), s^t_{j,f} = 1, \forall v_j \in V \bigcup c\} \tag{7}$$

3.4 Caching Benefit

As discussed in Section 2, a user retrieves data from a local edge server covering it or a remote edge server caching the data as long as it does not violate the app-specific latency constraint Φ_L. Thus, caching benefit is yielded when a user can retrieve data in the MEC system within Φ_L. Let $\mathcal{B}^t_{m,f}$ be the caching benefit yielded for individual user u_m's retrieval of data d_f. It can be calculated as follows:

$$\mathcal{B}^t_{m,f} = \begin{cases} \max\{\Phi_L - \Phi^t_{m,f}, 0\} & \text{if } u_m \text{ retrieves } d_f \text{ from an edge server} \\ 0 & \text{if } u_m \text{ retrieves } d_f \text{ from the remote cloud} \end{cases} \tag{8}$$

Now, the caching benefit yield by an MEDC strategy in time slot t can be calculated by:

$$\mathcal{B}(s^t) = \sum_{u_m \in U} \sum_{d_f \in D} \mathcal{B}^t_{m,f} \cdot q^t_{m,f} \tag{9}$$

3.5 Problem Formulation And Hardness

Let γ denote the app vendor's priority for lowering users' data retrieval latency. A large γ indicates that the app vendor is inclined to lower network latency for its users at higher caching costs, and vice versa. Based on the cost and benefit models in Sections 3.3 and 3.4, the caching revenue produced by MEC strategy s^t, denoted by $\mathcal{P}(s^t)$, can be calculated by caching benefit $\mathcal{B}(s^t)$ minus caching cost $\mathcal{C}(s^t)$:

$$\begin{aligned} \mathcal{P}(s^t) = \gamma \cdot \mathcal{B}(s^t) - \mathcal{C}(s^t) = \sum_{u_m \in U} \sum_{d_f \in D} \gamma \cdot \mathcal{B}^t_{m,f} \cdot q^t_{m,f} \\ - \sum_{v_i \in V} \sum_{d_f \in D} |d_f| \cdot s^t_{i,f} \cdot (1 - s^{t-1}_{i,f}) \cdot cost^t_{i,f} \end{aligned} \tag{10}$$

Now, the MEDC problem over a period of time T that consists of multiple time slots can be modeled as follows:

$$\begin{aligned} \max \quad & \lim_{T \to \infty} \sum_{t=1}^{T} \mathcal{P}(s^t) \\ s.t.: \quad & (1), (2), (3), (4) \end{aligned}$$

Now, we prove the $\mathcal{NP}$-hardness of the MEDC problem in an individual time slot t (referred to as the t-MEDC problem hereafter) with Theorem 1.

Theorem 1. *The t-MEDC problem in a time slot is $\mathcal{NP}$-hard.*

Proof To do the proof, we first introduce the weighted k-set packing (WKSP) problem, a classic $\mathcal{NP}$-hard problem. Let $\mathcal{X}$ denote an element universe with $\forall x \in \mathcal{X}$ and S denote the set of all subsets in $\mathcal{X}$. Each subset $s \in S$ covers a set of elements with weight $w(s)$. Given the limit k, the WKSP problem can be represented by:

$$\max \sum_{s \in S} w(s) \cdot c_s \tag{11a}$$

$$s.t.: \sum_{s \in S} c_s \leq k \tag{11b}$$

$$\sum_{x \in \mathcal{X}} c_x \leq 1 \tag{11c}$$

where $c_s \in \{0, 1\}$ indicates whether set $s \in S$ is included into the solution, and $c_x \in \{0, 1\}$ indicates whether element x is covered.

Now we present how to reduce a special case of the t-MEDC problem with same size data to the WKSP problem. Here we construct an 1-time-slot instance $t - MEDC(V, U, n, \mathcal{P}(s))$ based on a given instance $MEDC(V, U, n, \mathcal{P}(s))$ in the polynomial time where $|S| = |V|$, $|\mathcal{X}| = |U|$ and $n = k$. The function $\mathcal{P}(s)$ is the caching revenue produced by the MEDC strategy s based on (10). In this case, we can project the revenue function $\mathcal{P}(s)$ to $w(s)$. This way, any solution satisfying the objective of the t-MEDC problem also satisfies objective (11a). As constraint (11b) constrains the maximum number of selected sets, $\sum_{v_i \in V} \sum_{d_f \in D} s_{i,f} \leq \sum_{v_i \in V} \mathcal{A}_i$ based on (3) can be projected to (11b). Since the caching benefit of each data request d_f of user u_m can only be counted once, (11c) is satisfied. In this case, any solution satisfying constraints (3) and (4) also satisfies constraints (11b) and (11c). Thus, the WKSP problem can be reduced from the t-MEDC problem. Since the WKSP is $\mathcal{NP}$-hard, the t-MEDC problem is also $\mathcal{NP}$-hard. □

The t-MEDC problem is a special case of the MEDC problem where $T = 1$. Thus, the MEDC problem over T is also $\mathcal{NP}$-hard based on Theorem 1.

Algorithm 1 Single Time-slot MEDC Algorithm (ST-MEDC)

1: initialization
2: $\tilde{s}, \hat{s} \leftarrow \varnothing$
3: end of initialization
4: $\tilde{s}_{i,f}$ $\varnothing$
5: **repeat**
6: $\tilde{s}$ $\tilde{s} \bigcup \tilde{s}_{i,f}$
7: obtain the most cost-effective data caching decision $\tilde{s}_{i,f}$ under cache space constraint: $\tilde{s}_{i,f} = \arg\max\{\frac{\mathcal{B}(\tilde{s} \bigcup \tilde{s}_{i,f}) - \mathcal{B}(\tilde{s})}{|d_f|}, \forall v_i \in V, d_f \in D\}$
8: **until** no feasible decision $\tilde{s}_{i,f}$
9: $\hat{s}_{i,f}$ $\varnothing$
10: **repeat**
11: $\hat{s}$ $\hat{s} \bigcup \hat{s}_{i,f}$
12: obtain the data caching decision $\hat{s}_{i,f}$ with the highest benefit increase under cache space constraint: $\hat{s}_{i,f} = \arg\max\{\mathcal{B}(\hat{s} \bigcup \hat{s}_{i,f}) - \mathcal{B}(\hat{s}), \forall v_i \in V, d_f \in D\}$
13: **until** no feasible decision $\hat{s}_{i,f}$
14: **return** $\arg\max_{s \in \{\tilde{s}, \hat{s}\}} \mathcal{B}(s)$

4 Algorithm Design and Analysis

The complete information about the MEC system over time is required to solve the MEDC problem optimally. However, it is unrealistically in real-world scenarios where users' data requests usually arrive dynamically. In this section, we first present the ST-MEDC (Single Time-slot MEDC) algorithm for solving the MEDC problem in a single time slot. Then, we introduce the OL-MEDC (Online MEDC) algorithm for finding near-optimal MEDC strategies over time.

4.1 Single Time-slot MEDC Algorithm

Finding optimal solutions in large-scale MEDC scenarios is intractable, due to the $\mathcal{NP}$-hardness of the t-MEDC problem. Thus, OL-MEDC employs a heuristic algorithm named ST-MEDC (Single Time-slot MEDC) to solve the t-MEDC problem, aiming to maximize the caching benefit $\mathcal{B}(s^t)$ in individual time slots.

The pseudo-code of ST-MEDC is shown in Algorithm 1. Starting with initialization, ST-MEDC creates two initial MEDC strategy candidates $\tilde{s}$ and $\hat{s}$ (Lines 1-3). Then, it obtains $\tilde{s}$ first in an iterative manner: always including the data caching decision $\tilde{s}_{i,f}$ that produces the highest ratio of caching benefit over used caching spaces $\frac{\mathcal{B}(\tilde{s} \bigcup \tilde{s}_{i,f}) - \mathcal{B}(\tilde{s})}{|d_f|}$ into $\tilde{s}$ until the cache space constraint (3) is violated or no feasible decision $\tilde{s}_{i,f}$ can be found (Lines 4-8). Similarly, the algorithm obtains $\hat{s}$ by selecting the data caching decision $\hat{s}_{i,f}$ with the highest increase in caching benefit $\mathcal{B}(\tilde{s} \bigcup \tilde{s}_{i,f}) - \mathcal{B}(\tilde{s})$ (Lines 9-13). In Line 14, the solution with the higher total caching benefit is the final result of Algorithm 1.

The computational complexities of finding decisions in Lines 7 and 12 are at most $O(|V| \cdot |D|)$, respectively. There are $\sum_{v_i \in V} \mathcal{A}_i$ at most iterations in each of the two loops in ST-MEDC. Thus, the computational complexity of ST-MEDC is $O(2 \cdot |V| \cdot |D| \sum_{v_i \in V} \mathcal{A}_i) = O(|V|^2 \cdot |D|)$.

Algorithm 2 Online MEDC Algorithm (OL-MEDC)

1: initialization
2: $\beta = 0, t = 1, s^0 \leftarrow \varnothing, \mathcal{S} = \{s^0\}$
3: end of initialization
4: **while** $t \leq T$ **do**
5: obtain MEDC strategy s^t by Algorithm 1
6: calculate $\mathcal{B}(s^t), \mathcal{C}(s^t)$ by s^t and s^{t-1}
7: calculate $\mathcal{B}(s^{t-1})$ by s^{t-1}
8: **if** $\gamma \cdot \beta \geq k \cdot \mathcal{C}(s^t)$ **or** $\beta = 0$ **then**
9: $\beta = \mathcal{B}(s^t)$
10: **else**
11: $\beta = \beta + \mathcal{B}(s^{t-1})$
12: s^t s^{t-1}
13: **end if**
14: $\mathcal{S}$ $\mathcal{S} \bigcup s^t$
15: $t = t + 1$
16: **end while**

4.2 Online MEDC Algorithm

Now we present OL-MEDC, the online algorithm for formulating cost-effective MEDC strategies over time based on ST-MEDC. The pseudo-code of OL-MEDC is shown in Algorithm 2. As discussed in Section 2 and Section 3, caching cost is an important component in the caching revenue. Updating the MEDC strategy in every time slot may incur high caching costs. Thus, OL-MEDC updates the current MEDC strategy only when it has already produced adequate benefits, i.e. $\gamma \cdot \beta > k$ times the caching cost incurred by implementing the MEDC strategy update, where β is the accumulated caching benefit produced by the current MEDC strategy since its implementation and k is a parameter specified by the app vendor based on its willingness to trade off caching cost for caching benefit. In general, a large k will tend to reduce caching costs by keeping the current MEDC strategy.

Algorithm 2 initializes the accumulated caching benefit by $\beta = 0$ and creates an initial MEDC strategy (Lines 1-3). In each time slot, Algorithm 2 first obtains an approximation solution s^t with Algorithm 1 (Line 5), then calculates the benefit produced by s^t and the caching cost incurred by updating s^{t-1} to s^t (Line 6). After that, it calculates the caching benefit produced with the current MEDC strategy unchanged (Line 7). Then, through Line 8 to Line 15, the algorithm compares the accumulated caching benefit β obtained by s^{t-1}: if $\beta > \frac{1}{\gamma}$ times $\mathcal{C}(s^t)$ or s^{t-1} is infeasible, s^{t-1} is updated by s^t. Otherwise, s^{t-1} remains and no extra caching cost incurs.

As discussed in Section 4.1, the computational complexity of ST-MEDC is $O(|V|^2 \cdot |D|)$. Thus, the computational complexity of OL-MEDC in each time slot is also $O(|V|^2 \cdot |D|)$. This indicates the high efficiency of OL-MEDC and allows it to formulate MEDC strategies rapidly over time.

4.3 Performance Analysis

Here, we first analyze the approximation ratio of ST-MEDC in terms of caching benefit, i.e., the ratio of the caching benefit produced by ST-MEDC in the worst case over that produced by the optimal solution. After that, we analyze

the competitive ratio of OL-MEDC over time, i.e., the ratio of the total caching revenue produced by OL-MEDC in the worst case over that produced by the optimal strategy, based on the approximation ratio of ST-MEDC.

In terms of caching benefit, we obtain the approximation ratio of ST-MEDC by analyzing the performance bound of $\tilde{s}$ with Theorems 2 - 5. We divide each data cache decision $\tilde{s}_{i,f}$ into $|d_f|$ sub-decisions, where each sub-decision produces $\frac{\mathcal{B}(\tilde{s}\bigcup\tilde{s}_{i,f})-\mathcal{B}(\tilde{s})}{|d_f|}$ caching benefit. Let $\tilde{s}'$ denote the set of sub-decisions based on data caching strategy $\tilde{s}$.

Let $\tilde{s}'_l$ denote the sub-decision set when the l^{th} sub-decision is included in $\tilde{s}'$. Let s^* denote the optimal solution of the t-MEDC problem, and $\mathcal{B}(s^*)$ denote the caching benefit yielded by the optimal t-MEDC strategy. The increase in the caching benefit by including the l^{th} sub-decision, denoted by $\Delta\tilde{\mathcal{B}}_l$, is at least $\frac{\mathcal{B}(s^*)-\mathcal{B}(\tilde{s}'_{l-1})}{\sum_{v_i\in V}\mathcal{A}_i}$.

Theorem 2. For the l^{th} sub-decision included in $\tilde{s}'$, the increase in benefit, $\Delta\tilde{\mathcal{B}}_l$, follows:

$$\Delta\tilde{\mathcal{B}}_l \geq \frac{\mathcal{B}(s^*)-\mathcal{B}(\tilde{s}'_{l-1})}{\sum_{v_i\in V}\mathcal{A}_i} \tag{12}$$

Proof To proof this theorem, we first divide the optimal the optimal t-MEDC strategy s^* into a set of sub-decisions s'^*, where $\mathcal{B}(s^*)=\mathcal{B}(s'^*)$. Since $\tilde{s}'_l$ selects the sub-decision with the maximum ratio of benefit when including the lth sub-decision, for the first sub-decision in s'^* but not in $\tilde{s}'_{l-1}$, it is at most $\Delta\tilde{\mathcal{B}}_l$ increased in the caching benefit. In addition, there is at most $\sum_{v_i\in V}\mathcal{A}_i$ cache spaces available in the MEC system. Thus, the total caching benefit produced by $\{s'|s'\in s'^*\bigcap\neg\tilde{s}'_{l-1}\}$ is at most $\sum_{v_i\in V}\mathcal{A}_i\cdot\Delta\tilde{\mathcal{B}}_l$. Thus, the above inequality is satisfied. □

Now, we prove that the caching benefit produced by $\tilde{s}'_l$ provided by ST-MEDC with the l^{th} sub-decision included, is at least $\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l-1}\right)$ times the caching benefit produced by s'^* with Theorem 3.

Theorem 3. For each included sub-decision l, the caching benefit produced by $\tilde{s}'_l$ fulfills the following:

$$\mathcal{B}(\tilde{s}'_l)\geq\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l-1}\right)\mathcal{B}(s'^*) \tag{13}$$

Proof According to Theorem 2, we can obtain the caching benefit produced by $\tilde{s}'_l$ based on (14).

$$\begin{aligned}\mathcal{B}(\tilde{s}'_l)&=\mathcal{B}(\tilde{s}'_{l-1})+\Delta\tilde{\mathcal{B}}_l\geq\mathcal{B}(\tilde{s}'_{l-1})+\frac{\mathcal{B}(s'^*)-\mathcal{B}(\tilde{s}'_{l-1})}{\sum_{v_i\in V}\mathcal{A}_i}\\&=\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)\mathcal{B}(\tilde{s}'_{l-1})+\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\mathcal{B}(s'^*)\end{aligned} \tag{14}$$

This way, we can prove this theorem by the inductive proof easily, and omitted details here. □

According to Theorem 3, the lower bound of the caching benefit with the $(l+1)^{th}$ sub-decision of $\tilde{s}'$ can be calculated:

$$\mathcal{B}(\tilde{s}'_{l+1})\geq\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l}\right)\mathcal{B}(s'^*) \tag{15}$$

The total amount of available cache spaces is $\sum_{v_i\in V}\mathcal{A}_i$. When all the cache spaces are occupied ($l=\sum_{v_i\in V}\mathcal{A}_i$), the caching benefit produced by strategy $\tilde{s}_{l+1}$ fulfils:

$$\begin{aligned}\mathcal{B}(\tilde{s}'_{l+1})&\geq\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l}\right)\mathcal{B}(s'^*)\\&=\left(1-\left(1-\frac{1}{l}\right)^{l}\right)\mathcal{B}(s'^*)\geq\left(1-\frac{1}{e}\right)\mathcal{B}(s'^*)\end{aligned} \tag{16}$$

However, $\tilde{s}_{l+1}$ violates the cache space constraint with the $(l+1)^{th}$ sub-decision included into $\tilde{s}'_{l+1}$, where $l=\sum_{v_i\in V}\mathcal{A}_i$. Here, we prove the lower bound of the caching benefit produced by $\tilde{s}'_l$ with Theorem 4.

Theorem 4. When $l=\sum_{v_i\in V}\mathcal{A}_i$, the benefit produced by $\tilde{s}'_l$ is at least $\frac{e-1}{2e}$ times what is produced by s'^*:

$$\mathcal{B}(\tilde{s}'_l)=\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})\geq\frac{e-1}{2e}\mathcal{B}(s'^*)$$

Proof According to (16), the benefit produced by $\tilde{s}'_l$ fulfills:

$$\mathcal{B}(\tilde{s}'_l)\geq\left(1-\frac{1}{e}\right)\mathcal{B}(s'^*)-\tilde{\Delta\mathcal{B}_{l+1}} \tag{17}$$

Since the increase in the caching benefit in the $l+1^{th}$ time slot $\tilde{\Delta\mathcal{B}_{l+1}}$ cannot be higher than that in the l^{th} time slot:

$$\tilde{\Delta\mathcal{B}_{l+1}}\leq\Delta\tilde{\mathcal{B}}_l\leq\mathcal{B}(\tilde{s}'_l) \tag{18}$$

Thus, we can obtain:

$$\mathcal{B}(\tilde{s}'_l)\geq\left(1-\frac{1}{e}\right)\mathcal{B}(s'^*)-\mathcal{B}(\tilde{s}'_l)\geq\frac{e-1}{2e}\mathcal{B}(s'^*) \tag{19}$$

Thus, the theorem holds. □

Please note that the MEDC strategy $\tilde{s}$ obtained by ST-MEDC considers the differentiated data sizes. Thus, the set of sub-decisions $\tilde{s}'$ derived from $\tilde{s}$ will not always contain up to $\sum_{v_i\in V}\mathcal{A}_i$ sub-decisions. Now, we analyze the approximation ratio of the t-MEDC strategy s provided by ST-MEDC in terms of the caching benefit:

Theorem 5. The caching benefit produced by ST-MEDC is at least $\frac{(1-\omega)(e-1)}{2e}$ times the caching benefit produced by the optimal solution s^*, where $\omega=\frac{|V|\cdot\min\left\{\min\left\{\mathcal{A}_i,\forall v_i\in V\right\},\max\left\{|d_f|,\forall d_f\in D\right\}\right\}}{\sum_{v_i\in V}\mathcal{A}_i}$.

Proof We first analyze the performance of $\tilde{s}$ in ST-MEDC here, to obtain the approximation ratio of ST-MEDC.

$$\begin{aligned}\mathcal{B}(\tilde{s})&\geq\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})-\omega\cdot\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})\\&=(1-\omega)\cdot\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})\geq(1-\omega)\cdot\frac{e-1}{2e}\mathcal{B}(s'^*)\\&=\frac{(1-\omega)(e-1)}{2e}\cdot\mathcal{B}(s'^*)=\frac{(1-\omega)(e-1)}{2e}\cdot\mathcal{B}(s^*)\end{aligned} \tag{20}$$

Since the final MEDC strategy constituted by ST-MEDC always produces benefit no lower than $\tilde{s}$, we can obtain:

$$\mathcal{B}(s)=\max\{\mathcal{B}(\tilde{s}),\mathcal{B}(\hat{s})\}\geq\mathcal{B}(\tilde{s})\geq\frac{(1-\omega)(e-1)}{2e}\cdot\mathcal{B}(s^*) \tag{21}$$

Thus, the approximation ratio of ST-MEDC in terms of caching benefit is $\frac{(1-\omega)(e-1)}{2e}$. □

According to Theorem 5, we can prove the competitive ratio of OL-MEDC in terms of caching revenue:

Theorem 6. The competitive ratio of OL-MEDC is $\frac{(k-\gamma)(1-\omega)^2(e-1)^2\eta}{4ke^2}$, where $\eta = \frac{\min\left\{\mathcal{B}(s^t),\forall t\in\{1,...,T\}\right\}}{\max\left\{\mathcal{B}(s^t),\forall t\in\{1,...,T\}\right\}}$.

Proof According to Line 8 in Algorithm 2, the caching cost is at most $\frac{\gamma}{k}$ times the accumulated caching benefit β. In the worst case, the MEDC has to be updated in every time slot over T and the total caching cost incurred ($\sum_{t=1}^{T}\mathcal{C}(s^t)$) is no more than $\frac{\gamma}{k}\sum_{t=1}^{T}\mathcal{B}(s^t)$. Thus, the total caching revenue produced by $\mathcal{S} = \{s^0, s^1, ..., s^T\}$ can be calculated by:

$$\sum_{t=1}^{T}\mathcal{P}(s^t) = \sum_{t=1}^{T}(\mathcal{B}(s^t) - \mathcal{C}(s^t)) \geq (\frac{k-\gamma}{k})\cdot\sum_{t=1}^{T}\mathcal{B}(s^t) \quad (22)$$

Let $\sum_{t=1}^{T}\mathcal{P}(s^{*t})$ denote the caching revenue obtained by the offline optimal approach over T.

$$\begin{aligned}\sum_{t=1}^{T}\mathcal{P}(s^t) &\geq (\frac{k-\gamma}{k})\cdot\sum_{t=1}^{T}\frac{(1-\omega)(e-1)}{2e}\mathcal{B}^*_{min}\\ &\geq (\frac{k-\gamma}{k})\cdot\frac{(1-\omega)(e-1)}{2e}\cdot\sum_{t=1}^{T}\eta^*\cdot\mathcal{B}^*_{max}\\ &\geq (\frac{k-\gamma}{k})\cdot\left(\frac{(1-\omega)(e-1)}{2e}\right)^2\cdot\eta\cdot\sum_{t=1}^{T}\mathcal{P}(s^{*t})\end{aligned} \quad (23)$$

Thus, OL-MEDC always provides a solution that achieves $\frac{(k-\gamma)(1-\omega)^2(e-1)^2\eta}{4ke^2}$ times the optimal offline solution in terms of the caching revenue.

□

5 Experimental Evaluation

5.1 Benchmark Approaches

In the experiments, OL-MEDC is compared with five representative approaches :

- **IP-MEDC**: In each individual time slot, it provides the optimal solution $\mathcal{P}(s^t)$ to t-MEDC problem described in Section 3.5 with IBM's CPLEX Optimizer.
- **CEDC-O** [3]: This online approach aims to maximize the coverage of user requests while minimizing the system cost. As mentioned in Section 2, the app vendor needs to reserve cache spaces. Thus, the cost of hiring cache spaces is not included in this approach.
- **Request-based Collaborative Caching (RCC)** [22]: This online approach focuses on serving the most users by the MEDC strategy over time in the MEC system.
- **AEDC** [23]: This offline approach finds solutions to t-MEDC problems with the aim to approximate the maximum caching benefits obtained by IP-MEDC. In the experiments, AEDC runs in each individual time slot to obtain the results.
- **Distributed Caching Algorithm (DCA)**: This distributed algorithm originates from [24] and is enhanced to solve the online MEDC problem. Edge servers communicate and cache data collaboratively in each time slot. This algorithm maximizes the overall caching revenue by heuristically caching data on edge servers that yield the highest caching revenues.

5.2 Experimental Settings

The real-world EUA dataset[1] is used for conducting the experiments in this study. In the experiments, a total of 200 mobile users are randomly selected from the dataset to simulate users in the system. According to the experimental settings, a certain number of edge servers are randomly selected from the dataset and connected to simulate an MEC system. The latency limit Φ_L is set to 2 hops. Similar to [25], a number of these users are randomly selected to send requests for a set of data (D) in each time slot, following $\mathcal{N}(\mu, \sigma^2)$, a normal distribution, where $\mu = \frac{|M|}{2}$ and $\sigma = \frac{|M|}{4}$. The sizes of data requested are also randomly selected between 1 to the maximum reserved cache spaces in the experiments.

AWS's Snowball Edge Pricing[2] is adopted in the experiments. We set c_{ce} to \$0.016, c_{ee} to \$0.006 and γ to \$0.004. The normal distribution $\mathcal{N}(\mu', \sigma'^2)$ is used to randomly generate the reserved cache spaces on every edge server, where μ' equals to half of the maximum reserved cache spaces among all the edge servers and $\sigma' = 1$.

5.3 Parameter Settings

To evaluate OL-MEDC comprehensively, we conduct seven sets of experiments to simulate various MEDC scenarios. Set #1 aims to demonstrate and compare the performance of the six approaches over 100 time slots, i.e., $T = 100$. Set #2 aims to demonstrate the applicability of OL-MEDC in four different MEDC modes:

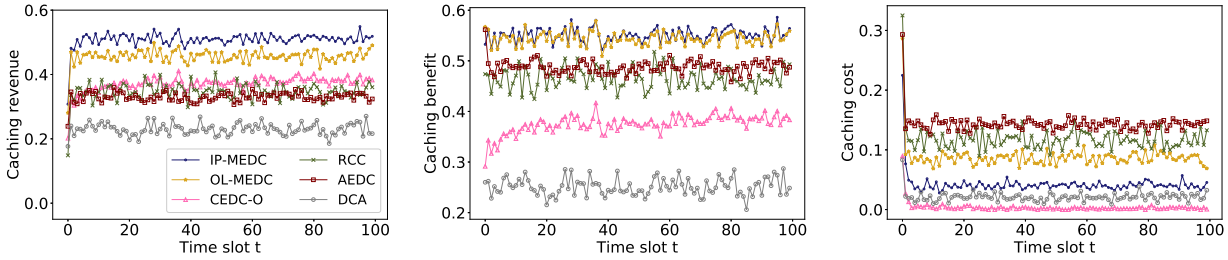
- **General Mode (GM)**. In this mode, the generic latency and cost models presented in Section 3 are applied and data demands in individual time slots follow a discrete uniform distribution across individual edge servers.
- **Zipf Mode (ZM)**. In this mode, the generic latency and cost models are applied in the same way as GM, while users' demands for different data follow a Zipf distribution, similar to [26].
- **Latency-specific Mode (LM)**. In this mode, a specific latency model is applied by randomly selecting a latency value from $(0, 2)$ for each link between two edge servers. In addition, a generic cost model is applied and data demands follow a discrete uniform distribution.
- **Cost-specific Mode (CM)**. In this mode, a specific cost model is applied by randomly selecting a cost value from $(0, 2)$ for each link between two edge servers. In addition, the generic latency model is applied and data demands follow a discrete uniform distribution.

As summarized in Table 2, we vary the value of one of the following five parameters while fixing the others in Sets #2-#7. In this way, we can evaluate OL-MEDC in different MEDC scenarios and demonstrate the impacts of the parameters. In these sets, each experiment also continues

1. https://github.com/swinedge/eua-dataset
2. https://aws.amazon.com/snowball-edge/pricing/

TABLE 2: Parameter Settings

	$Mode$	$\|V\|$	θ	MS	$\|D\|$	k
Set #1	GM	10	1.0	4	4	1
Set #2	GM, TM, LM, CM	10	1.0	4	4	1
Set #3	GM	6, 8, 10, 12, 14, 16	1.0	4	4	1
Set #4	GM	10	1.0, 1.2, 1.4, 1.6, 1.8, 2.0	4	4	1
Set #5	GM	10	1.0	2, 3, 4, 5, 6	4	1
Set #6	GM	10	1.0	4	2, 3, 4, 5, 6	1
Set #7	GM	10	1.0	4	4	1, 4, 16, 64, 256



(a) $\mathcal{P}(s^t)$ vs. t (b) $\mathcal{B}(s^t)$ vs. t (c) $\mathcal{C}(s^t)$ vs. t

Fig. 2: Set #1

for 100 time slots, is repeated 100 times when a setting parameter varies, and the average results are reported. The changed parameters in Sets #3-#7 are:

- Number of edge servers ($|V|$). This parameter varies from 6 to 16 in steps of 2 and impacts the size of the MEC system.
- Edge density (θ). Given n edge servers in the simulated MEC system, e edges are randomly generated based on $\theta = e/n$. This parameter increases from 1 to 2 in steps of 0.2.
- Maximum reserved cache spaces among all edge servers (MS). This parameter decides the maximum reserved cache spaces on the edge servers, varying from 2 to 6 in steps of 1.
- Number of data ($|D|$). This parameter increases from 2 to 6 in steps of 1 and is the number of users' requested data over T.
- Parameter k in Algorithm 2. This parameter varies among 1, 4, 16, 64, 256 and is used in Algorithm 2 to determine the updating frequency of the data caching strategy.

As discussed in Section 2, it is cost-ineffective to reserve huge cache spaces on individual edge servers. Therefore, those reserved cache spaces must not exceed the threshold MS in the experiments.

5.4 Performance Metrics

Four performance metrics are adopted in the experiments for evaluating OL-MEDC:

- Caching revenue $\mathcal{P}(s^t)$, the higher the better.
- Caching benefit $\mathcal{B}(s^t)$, the higher the better.
- Caching cost $\mathcal{C}(s^t)$, the lower the better.
- Maximum computation time, measured by seconds, the lower the better.

In the evaluation, we observe the maximum computation time of an approach across the 100 time slots to measure its efficiency and feasibility, rather than the average computation time. This is because if the approach freezes in any of the time slots due to excessive computation time, it will not be able to continue to update the MEDC strategy for the rest of T. Since DCA is a distributed algorithm, the computation time in each time slot is determined by the most time-consuming data caching decision. Please note that the efficiency results of Set #7 is not presented because k does not impact the computation time of OL-MEDC.

5.5 Effectiveness

Figs. 2 - 8 show the experimental results of all seven sets of experiments. Overall, *IP-MEDC achieves the highest average caching revenue, closely followed by OL-MEDC*. The average advantages of IP-MEDC and OL-MEDC are 29.78% and 20.41% over CEDC-O, 39.52% and 29.45% over RCC, 40.05% and 29.94% over AEDC and 149.80% and 131.76% over DCA.

Fig. 2 depicts the results of Set #1. Overall, *IP-MEDC and OL-MEDC's performance are stable over time, outperforming the other four approaches significantly in maximizing the caching revenue.* Fig. 2(a) shows that the average caching revenues achieved by IP-MEDC and OL-MEDC are 37.09% and 22.96% higher than CEDC-O, 46.64% and 31.52% higher than RCC, 54.36% and 38.45% higher than AEDC, 121.30% and 98.49% higher than DCA. The average caching revenue achieved by OL-MEDC reaches 89.69% of that achieved by IP-MEDC. This indicates the high effectiveness of OL-MEDC. The advantages of IP-MEDC and OL-MEDC in maximizing caching revenue come from their high ability to achieve high caching benefits. This can be seen in Fig. 2(b). Achieving comparable caching benefits, IP-MEDC and

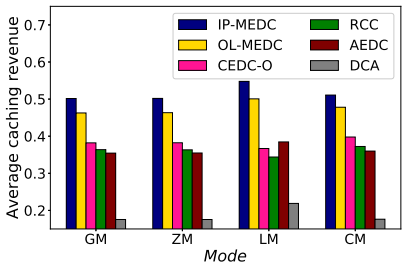


Fig. 3: Set #2

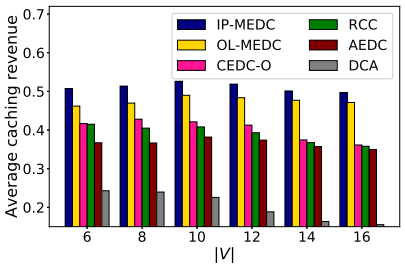


Fig. 4: Set #3

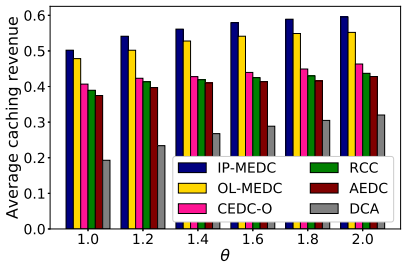


Fig. 5: Set #4

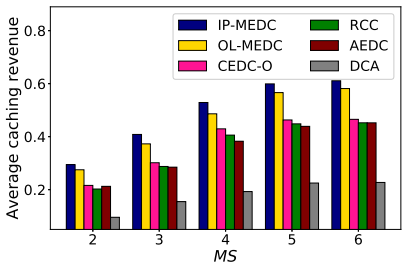


Fig. 6: Set #5

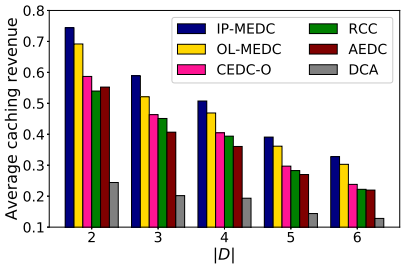


Fig. 7: Set #6

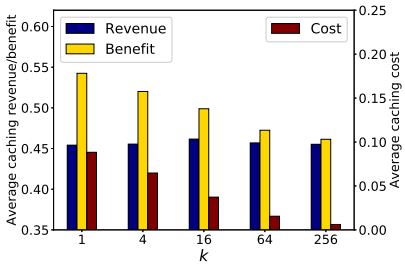


Fig. 8: Set #7

OL-MEDC outperform CEDC-O, RCC, AEDC and DCA by large margins. Specifically, their average caching benefits are 46.98% and 45.17% higher than CDEC-O, 18.29% and 16.83% higher than RCC, 12.95% and 11.56% higher than AEDC, and 119.47% and 116.78% higher than DCA. Fig. 2(c) compares the caching costs incurred by the six approaches over T. CEDC-O incurs the lowest caching cost because CEDC-O aims to minimize the system cost. On average, the caching costs incurred by IP-MEDC and OL-MEDC are 0.0424 and 0.0881, lower than RCC and AEDC's 0.1190, 0.1583, but higher than CEDC-O and DCA's 0.0039 and 0.0212.

Fig. 3 demonstrates the average caching revenues achieved by all six approaches in Set #2. *In all four modes, IP-MEDC and OL-MEDC outperform CEDC-O, RCC, AEDC and DCA with large margins*. The average advantages of IP-MEDC and OL-MEDC are 34.87% and 24.58% over CEDC-O, 42.88% and 31.97% over RCC, 41.86% and 31.03% over AEDC and 176.39% and 155.30% over DCA. In the ZM mode where data demands follow the Zipf distribution, the six approaches achieve caching revenues similar to those achieved in other modes. This indicates the ability of OL-MEDC to accommodate data demands following different patterns. The average caching revenues achieved by IP-MEDC and OL-MEDC in the LM and CM modes are higher than those in GM and ZM modes. In the LM mode, IP-MEDC and OL-MEDC can effectively increase caching benefits by delivering data through low-latency links, which increases caching revenues. In the CM mode, they excel at minimizing caching costs by delivering data at low transmission costs, which also increases caching revenues.

Fig. 4 - Fig. 7 show the experimental results of Set #3 - #6. Those figures demonstrate that *IP-MEDC and OL-MEDC significantly outperform the other four approaches again* in terms of caching revenue. The results of Set #3 is depicted in Fig. 4 with various numbers of edge servers. With the initial increase from 6 edge servers, the caching revenues achieved by IP-MEDC and OL-MEDC increase. The reason is that more users can be served by their nearby edge servers and the caching benefits increases correspondingly. However, the caching revenues decrease when the number of edge servers exceeds 10. This is because the maximum achievable caching benefits are fixed with the fixed numbers of users. With more edge servers, there are more cache spaces in the MEC system. This increases the possibility of data transmissions, which potentially incurs extra caching costs. Thus, the caching revenues achieved by all approaches decrease in Fig. 4.

When the edge density θ increases, the caching revenue increases in Fig. 5, because each end-user has a higher chance of being served by an edge server under the latency constraint. As a result, all the approaches can achieve higher caching revenues. When the maximum reserved spaces increase from 2 to 6, the caching revenues achieved by all the fix approaches increase in Fig. 6. However, when the maximum reserved spaces increase from 5 to 6, the increase becomes much slower. This indicates that reserving a large amount of cache spaces on an individual edge server is usually cost-ineffective. Fig. 7 depicts the results of Set #6. When the number of requested data $|D|$ decreases, the caching revenues achieved by all the approaches decrease. The increase in $|D|$ leads to a higher possibility of transmitting data from the cloud because the reserved caching spaces are fixed. This way, it decreases the caching benefits and consequently the caching revenues.

Fig. 8 shows the impact of parameter k on OL-MEDC in caching revenue, benefit and cost. As discussed in Section 4.2, Algorithm 2 employs k to reduce caching costs. As shown in Fig. 8, *a larger k can indeed lower the caching cost*. However, it also decreases the caching benefit in the mean-

TABLE 3: Maximum computation time

	Set #1	Set #2	Set #3	Set #4	Set #5	Set #6
IP-MEDC	2.6540	2.6637	379.9838	3.3022	19.9294	7.6132
OL-MEDC	0.0095	0.0108	0.1541	0.0102	0.0087	0.0090
CDEC-O	0.3661	0.5285	3.1908	0.3836	0.6719	0.7918
RCC	0.0054	0.0061	0.0107	0.0078	0.0063	0.0075
AEDC	0.0057	0.0068	0.0129	0.0058	0.0066	0.0077
DCA	0.0005	0.0010	0.0008	0.0006	0.0005	0.0010

time. Thus, an app vendor pursuing maximum caching benefit despite caching cost can feed a small k value to Algorithm 2. If an app vendor wants to maximize the cost-effectiveness of its MEDC strategy, it can select a k value that maximizes the caching revenue, e.g., $k = 16$ in Set #7.

5.6 Efficiency

In the experiments, we use the maximum computation time taken across the 100 time slots to evaluate the efficiency. The results are presented in Table 3. It demonstrates that *IP-MEDC takes much more computation time than others*, due to the $\mathcal{NP}$-hardness of the t-MEDC problem. In particular, the maximum computation times taken by IP-MEDC and CDEC-O are 379.9838 seconds and 3.1908 seconds in Set #5, while the maximum computation times taken by OL-MEDC, RCC, AEDC and DCA are at most 0.1541 seconds, 0.0107 seconds, 0.0129 seconds and 0.0008 seconds, respectively. This shows that *OL-MEDC is computationally feasible to deploy on large scales, real-world edge data caching problems*.

5.7 Threats to Validity

In this section, we analyze the threats to the validity of the experimental evaluation, including the threats to construct validity, internal validity and external validity.

5.7.1 Threats to Construct Validity

The main threats to construct validity in the experiments are the generated graphs and five comparison approaches. Randomly generated graphs may not always illustrate real-world scenarios precisely. To minimize this threat, the graphs are randomly generated in each execution - 100 graphs are generated when a parameter changes. Moreover, the five comparison approaches, i.e., IP-MEDC, CDEC-O, RCC, AEDC and DCA, may not suffice to evaluate OL-MEDC comprehensively. To minimize this threat, we simulate different MEDC scenarios by varying five parameters. In addition, we also evaluate the applicability of OL-MEDC in different real-world scenarios by evaluating its performance in four MEDC modes.

5.7.2 Threats to Internal Validity

For the internal validity, the main threat is the experiment settings that may favor OL-MEDC over other approaches. To tackle this threat, we simulated various MEDC scenarios by changing six parameters for comprehensively and fairly comparing the performance of all the six approaches. In addition, the experiment was repeated 100 times to obtain the averaged results when a setting parameter varies. In this way, biased results obtained in extreme experiments, e.g., those with unrealistic data request distribution or edge server distribution, are neutralized.

5.7.3 Threats to External Validity

The generalize application of OL-MEDC in different MEDC scenarios is the main threat to the external validity. In this paper, we generically modeled the MEDC problem and evaluated all approaches to reduce this threat. We employed the number of hops to measure latency. Therefore, we can easily interpret the evaluation results with specific retrieval latency and data size models. Furthermore, the experiments were conducted on a real-world dataset. In addition, we changed the complexity and size of the MEDC problem by varying the parameters in the experiment settings. In this way, we can ensure the representativeness and comprehensiveness of experimental evaluations. Thus, the threat to external validity is mitigated.

6 Related Work

Mobile edge computing (MEC) extends cloud computing by pushing computing resources and services to the network edge [27]. App vendors can deploy their services and data on edge servers to offer their users low service latency and data retrieval latency. In the meanwhile, many new challenges are posed, including edge user allocation [28], edge application deployment [29], edge data integrity [30], edge DDoS mitigation [21], collaborative edge computing [20] and edge data caching [3], [31].

Existing data caching approaches for cloud computing and conventional distributed computing are rendered obsolete, due to the characteristics of MEC systems. Researchers are starting to study data caching problems in MEC systems. In [32], the authors considered the caching revenue produced during the data delivery process. They proposed an auction mechanism to find an optimal data caching solution. The authors of [33] proposed a new edge cache architecture by including caches on smart vehicles into the network caches. This approach improves the resource utility and the effectiveness of this architecture. However, these studies focus on offline approaches, which require complete information about the MEC system over time. Thus, dynamic MEDC scenarios cannot be handled by these offline approaches.

Very recently, a number of online data caching approaches have been proposed. The authors of [34] studied the joint service caching and task offloading problem in MEC. They proposed a Lyapunov-based approach, namely OREO, to achieve the minimum network latency while ensuring the low energy consumption in the long-term. Considering users' mobility, the authors of [35] propose MOREA that allocates various resources, i.e. cache spaces and CPU circles on edge servers for scheduling offloading tasks. The authors of [16] studied a micro-service deployment problem with the aim to minimize the overall cost instead of data retrieval latency. They proposed IDA4ReE, a primal-dual based algorithm, to find the solution to this problem with consideration of resource constraints and performance requirements. However, existing studies investigated data caching problems in the MEC systems

mainly to complement offloading scheduling. Thus, they failed to pay sufficient attention to data caching itself, as a unique technology aiming to reduce data retrieval latency, especially from the app vendor's perspective.

The MEDC problem was first studied from the app vendor's perspective in [31]. This research aims to cache data in an MEC system for serving all the users while minimizing the total cost. The main limitations of this study are the lack of consideration of edge servers' storage capacities and the data dynamics on MEC systems. In [36], Xia et al. considered cache space reservation in the MEDC problem. However, they ignored the dynamics in MEC systems, and only focused on caching benefit without considering caching cost. The authors of [3] provided a Lyapunov-based online algorithm, CEDC-O, for solving the MEDC problem dynamically. However, they unrealistically assumed that app vendors can always hire the needed storage spaces on edge servers without reservation, and focused solely on user coverage rather than caching revenue. The experiment results in Section 5.5 show that the caching revenue produced by CEDC-O is much lower than that produced by OL-MEDC. To the best of our knowledge, OL-MEDC is the first attempt to solve the MEDC problem efficiently for app vendors in an online manner, taking into the unique constraints and data dynamics in real-world MEC systems.

7 Conclusion

In this paper, we investigated the Mobile Edge Data Caching (MEDC) problem in MEC systems from the app vendor's perspective. We identified the major challenges and modeled the MEDC problem formally. We then proved that the MEDC problem is $\mathcal{NP}$-hard. To accommodate the dynamics of MEC systems, we proposed OL-MEDC, an online approach for formulating MEDC strategies over time with a provable performance guarantee. Through extensive experiments, the results demonstrated the advantages of OL-MEDC in maximizing the caching revenue in MEC systems, compared with representative approaches. We will consider MEC systems that allow data to be partitioned for caching in our future work.

Acknowledgement

This research is funded by Australian Research Council Discovery Projects No. DP180100212, DP200102491 and Laureate Fellowship FL190100035. Qiang He is the corresponding author of this paper.

References

[1] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[2] Z. Ning, P. Dong, X. Wang, X. Hu, L. Guo, B. Hu, Y. Guo, T. Qiu, and R. Y. K. Kwok, "Mobile edge computing enabled 5g health monitoring for internet of medical things: A decentralized game theoretic approach," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 463–478, 2021.

[3] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2020.

[4] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 514–522.

[5] G. Casale, "Analyzing replacement policies in list-based caches with non-uniform access costs," in *IEEE Conference on Computer Communications*. IEEE, 2018, pp. 432–440.

[6] A. Mukhopadhyay, N. Hegde, and M. Lelarge, "Optimal content replication and request matching in large caching systems," in *IEEE Conference on Computer Communications*, 2018, pp. 288–296.

[7] K. Ji, G. Quan, and J. Tan, "Asymptotic miss ratio of lru caching with consistent hashing," in *IEEE Conference on Computer Communications*, 2018, pp. 450–458.

[8] S. H. Lim, C.-Y. Wang, and M. Gastpar, "Information-theoretic caching: The multi-user case," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7018–7037, 2017.

[9] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, "Online resource allocation, content placement and request routing for cost-efficient edge caching in cloud radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1751–1767, 2018.

[10] S. Li and T. Lan, "Hotdedup: Managing hot data storage at network edge through optimal distributed deduplication," in *IEEE Conference on Computer Communications*. IEEE, 2020, pp. 247–256.

[11] J. Xie, D. Guo, X. Shi, H. Cai, C. Qian, and H. Chen, "A fast hybrid data sharing framework for hierarchical mobile edge computing," in *IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2609–2618.

[12] G. Luo, H. Zhou, N. Cheng, Q. Yuan, J. Li, F. Yang, and X. S. Shen, "Software defined cooperative data sharing in edge computing assisted 5g-vanet," *IEEE Transactions on Mobile Computing*, 2019.

[13] D. Malak, M. Al-Shalash, and J. G. Andrews, "Optimizing content caching to maximize the density of successful receptions in device-to-device networking," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4365–4380, 2016.

[14] F. Gabry, V. Bioglio, and I. Land, "On energy-efficient edge caching in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3288–3298, 2016.

[15] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "Distcache: Provable load balancing for large-scale storage systems with distributed caching," in *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, 2019, pp. 143–157.

[16] S. Deng, Z. Xiang, J. Taheri, K. A. Mohammad, J. Yin, A. Zomaya, and S. Dustdar, "Optimal application deployment in resource constrained distributed edges," *IEEE Transactions on Mobile Computing*, 2020.

[17] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 31–44, 2020.

[18] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[19] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 915–929, 2018.

[20] L. Yuan, Q. He, S. Tan, B. Li, J. Yu, F. Chen, H. Jin, and Y. Yang, "Coopedge: A decentralized blockchain-based platform for cooperative edge computing," in *Proceedings of the 30th Web Conference*, 2021.

[21] Q. He, C. Wang, G. Cui, B. Li, R. Zhou, Q. Zhou, Y. Xiang, H. Jin, and Y. Yang, "A game-theoretical approach for mitigating edge ddos attack," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[22] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1863–1876, 2016.

[23] X. Xia, F. Chen, Q. He, G. Cui, P. Lai, M. Abdelrazek, J. Grundy, and H. Jin, "Graph-based data caching optimization for edge computing," *Future generation computer systems*, vol. 113, pp. 228–239, 2020.

[24] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 852–864, 2019.

[25] S. Chen, L. Jiao, L. Wang, and F. Liu, "An online market mechanism for edge emergency demand response via cloudlet control," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2566–2574.

[26] Y. Yang and J. Zhu, "Write skew and zipf distribution: Evidence and implications," *ACM transactions on Storage (TOS)*, vol. 12, no. 4, pp. 1–19, 2016.

[27] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1459–1467.

[28] P. Lai, Q. He, G. Cui, F. Chen, J. Grundy, M. Abdelrazek, J. G. Hosking, and Y. Yang, "Cost-effective user allocation in 5g noma-based mobile edge computing systems," *IEEE Transactions on Mobile Computing*, 2021.

[29] B. Li, Q. He, G. Cui, X. Xia, F. Chen, H. Jin, and Y. Yang, "Read: Robustness-oriented edge application deployment in edge computing environment," *IEEE Transactions on Services Computing*, 2020.

[30] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing cache data integrity in the edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1210–1223, 2021.

[31] X. Xia, F. Chen, Q. He, G. Cui, P. Lai, M. Abdelrazek, J. Grundy, and H. Jin, "Graph-based optimal data caching in edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 477–493.

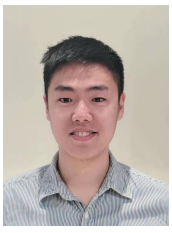
[32] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *38th IEEE International Conference on Distributed Computing Systems*, 2018, pp. 388–399.

[33] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5g networks with mobile edge computing," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 80–87, 2018.

[34] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE Conference on Computer Communications*, 2018, pp. 207–215.

[35] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "Moera: Mobility-agnostic online resource allocation for edge computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1843–1856, 2018.

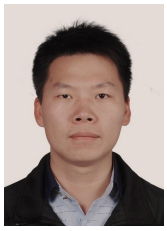
[36] X. Xia, F. Chen, J. Grundy, M. Abdelrazek, H. Jin, and Q. He, "Constrained app data caching over edge server graphs in edge computing environment," *IEEE Transactions on Services Computing*, 2021.



Xiaoyu Xia received his Master degree from The University of Melbourne, Australia in 2015. He is a PhD candidate at Deakin University. His research interests include edge computing, parallel and distributed computing, service computing, software engineering and cloud computing.

Feifei Chen received her PhD degree from Swinburne University of Technology, Australia in 2015. She is a lecturer at Deakin University. Her research interests include software engineering, cloud computing and green computing.

Qiang He received his first PhD degree from Swinburne University of Technology, Australia, in 2009 and his second PhD degree in computer science and engineering from Huazhong University of Science and Technology, China, in 2010. He is an Associate Professor at Swinburne. His research interests include service computing, software engineering, cloud computing and edge computing. More details about his research can be found at https://sites.google.com/site/heqiang/.



Guangming Cui received his Master degree from Anhui University, China, in 2018. He is a PhD candidate at Swinburne University of Technology. His research interests include software engineering, edge computing and service computing.

John C. Grundy received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently Australian Laureate Fellow and a professor of software engineering at Monash University, Melbourne, Australia. He is an Associate Editor in Chief of IEEE Transactions on Software Engineering, and Associate Editor of Automated Software Engineering Journal and IEEE Software. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at https://sites.google.com/site/johncgrundy/.

Mohamed Abdelrazek is an Associate Professor of Software Engineering and IoT at Deakin University. Before joining Deakin University in 2015, he worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). More details about his research can be found at https://sites.google.com/site/mohamedalmorsy/.

Athman Bouguettaya is a Professor in the School of Computer Science at University of Sydney, Australia. He received his PhD in Computer Science from the University of Colorado at Boulder (USA) in 1992. He is or has been on the editorial boards of several journals including, the IEEE Transactions on Services Computing, ACM Transactions on Internet Technology, the International Journal on Next Generation Computing and VLDB Journal. He was the recipient of several federally competitive grants in Australia (e.g., ARC) and the US (e.g., NSF, NIH). He is a Fellow of the IEEE and a Distinguished Scientist of the ACM.

Hai Jin received the Ph.D. degree in computer engineering from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1994. He is a Cheung Kung Scholars Chair Professor of computer science and engineering with the HUST. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

4.3 Summary

In this chapter, we attempt to solve the edge data replacement problems for app vendors, with consideration of dynamic data demands and user mobility, to answer **RQ2**. In **Chapter 4.1**, we formulate the online collaborative data replacement strategy named CEDC-O for app vendors in edge computing systems based on Lyapunov Theory, without requiring future information. However, the approach proposed in **Chapter 4.1** only considers the cost, and the benefit is also important to the success of app vendors. Thus, **Chapter 4.2** proposes a cost-effective online data replacement algorithm named OL-MEDC for app vendors to maximize the total revenue, including the cost and benefit. The experiment results show that OL-MEDC outperforms the state-of-the-art approaches including CEDC-O in both effectiveness and efficiency for addressing the online edge data replacement problem. This research has established the foundation for the edge data replacement problem and opened up many future research directions.

Chapter 5

Edge Data Distribution

Existing research efforts have focused on how to place and replace data across edge servers to achieve different optimization objectives. The fact that data transmission from within the cloud to distributed edge servers may incur excessive costs is largely ignored. For example, Amazon Web Services charges up to US$0.09 + US$0.02 to transfer 1GB data out of its S3 data storage facilities to the internet[1]. It is a significant component in the cost structure for app vendors to consider in the edge computing environment, similar to a large body of work on cloud computing [54, 63]. In this chapter, we attempt to answer **RQ3** by solving the edge data distribution problem, aiming to minimize the data distribution cost, in both quasi-static and dynamic edge computing scenarios based on a published paper and a submitted manuscript, respectively.

[1] https://aws.amazon.com/s3/pricing/

5.1 Cost-effective Data Distribution Strategies in Quasi-static Edge Computing Scenarios

Unlike data transmission in cloud computing and wireless sensor networks [31, 20, 41, 7], app data distribution in the edge computing environment consists of two major components: 1) data transmission from the cloud to edge servers; 2) and data transmission between edge servers. Both components must be considered in a systematic manner to formulate cost-effective data distribution strategies for app vendors. In this chapter, we study this edge data distribution problem in quasi-static edge computing scenarios, with the aim to minimize the data distribution cost while guaranteeing the data transmission latency. To solve this problem effectively and efficiently, we design an $O(k)$-approximation algorithm named EDD-A from the view of the Steiner Tree.

This chapter is based on a published paper, entitled: Cost-Effective App Data Distribution in Edge Computing, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32(1), pp. 31-44, 2020.

Cost-Effective App Data Distribution in Edge Computing

Xiaoyu Xia, Feifei Chen, Qiang He, *Member, IEEE*, John C. Grundy, *Senior Member, IEEE*, Mohamed Abdelrazek, and Hai Jin, *Fellow, IEEE*

Abstract—Edge computing, as an extension of cloud computing, distributes computing and storage resources from centralized cloud to distributed edge servers, to power a variety of applications demanding low latency, e.g., IoT services, virtual reality, real-time navigation, etc. From an app vendor's perspective, app data needs to be transferred from the cloud to specific edge servers in an area to serve the app users in the area. However, according to the pay-as-you-go business model, distributing a large amount of data from the cloud to edge servers can be expensive. The optimal data distribution strategy must minimize the cost incurred, which includes two major components, the cost of data transmission between the cloud to edge servers and the cost of data transmission between edge servers. In the meantime, the delay constraint must be fulfilled - the data distribution must not take too long. In this article, we make the first attempt to formulate this Edge Data Distribution (EDD) problem as a constrained optimization problem from the app vendor's perspective and prove its $\mathcal{NP}$-hardness. We propose an optimal approach named EDD-IP to solve this problem exactly with the Integer Programming technique. Then, we propose an $O(k)$-approximation algorithm named EDD-A for finding approximate solutions to large-scale EDD problems efficiently. EDD-IP and EDD-A are evaluated on a real-world dataset and the results demonstrate that they significantly outperform three representative approaches.

Index Terms—Edge computing, optimization, data distribution, cost-effectiveness, edge server network

1 Introduction

The world has witnessed exponentially growing mobile data traffic in this decade promoted by a huge increase in mobile devices and Internet of Things (IoT) connected devices [1]. This explosion of mobile data traffic has led to a wealth of research aiming to relieve the enormous data transmission loads on networks. Conventional network paradigms facilitated by cloud computing, including content delivery networks, content-centric networks and information centric networks, cannot handle the increases in network latency and network congestion caused by the rapidly increasing mobile traffic. In recent years, edge computing has emerged as a new computing paradigm that push computing and storage resources to the edge of the cloud [2]. These edge servers, each powered by one or many physical machines, are deployed at base stations or access points that are geographically close to devices [3]. Vendors of mobile and IoT applications (referred to as *app vendors* together hereafter) can hire computing and storage resources on edge servers for hosting their apps to serve their own app users with low latency [4]. Offloading computation tasks from app users' end-devices to nearby edge servers can ease the computation burden and energy consumption on those resource-limited end-devices [5], [6], [7], [8]. This is also a key technology of the 5G mobile network [9].

As edge servers become the entry points to the Internet for a larger number of mobile and IoT devices, a much larger proportion of the rapidly increasing mobile traffic data will be transmitted through those edge servers from the cloud. From an app vendor's perspective, caching app data on edge servers can considerably reduce the latency for their own users' data retrieval. In addition, it will largely reduce the volume of their app data transmitted between the cloud and its app users. This in turn will decrease the corresponding data transmission costs [10]. The new challenges raised by data caching in the edge computing environment have attracted many researchers' attention in recent years [11], [12], [13], [14], [15].

However, existing research efforts have focused on how to cache data across edge servers to achieve different optimization objectives, e.g., to minimize caching cost [14], to minimize retrieval latency [15], to guarantee the quality of transmissions [12], etc. The fact that data transmission from within the cloud to distributed edge servers may incur excessive costs is largely ignored. For example, Amazon Web Services charges up to US$0.09 + US$0.02 to transfer 1 GB data out of its S3 data storage facilities to the internet.[1] It

- *Xiaoyu Xia, Feifei Chen, and Mohamed Abdelrazek are with the School of Information Technology, Deakin University, Geelong, Victoria 3217, Australia. E-mail: {xiaoyu.xia, feifei.chen, mohamed.abdelrazek}@deakin.edu.au.*
- *Qiang He is with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria 3122, Australia. E-mail: qhe@swin.edu.au.*
- *John Grundy is with the Faculty of Information Technology, Monash University, Melbourne, Victoria 3800, Australia. E-mail: john.grundy@monash.edu.*
- *Hai Jin is with Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technolgoy, HuaZhong University of Science and Technology, Wuhan 430074, China. E-mail: hjin@hust.edu.cn.*

Manuscript received 28 Jan. 2020; revised 22 June 2020; accepted 16 July 2020. Date of publication 21 July 2020; date of current version 31 July 2020.
(Corresponding author: Qiang He.)
Recommended for acceptance by D. Medhi.
Digital Object Identifier no. 10.1109/TPDS.2020.3010521

1. https://aws.amazon.com/s3/pricing/

is a significant component in the cost structure for app vendors to consider in the edge computing environment, similar to a large body of work on cloud computing [16], [17]. Unlike data transmission in cloud computing and wireless sensor networks [18], [19], [20], [21], app data distribution in the edge computing environment consists of two major components: 1) data transmission from the cloud to edge servers; 2) and data transmission between edge servers. Both components must be considered in a systematic manner to formulate cost-effective data distribution strategies for app vendors. We refer to this problem as the edge data distribution (EDD) problem in this paper.

The scale of an EDD scenario can be very large and finding a solution is not trivial. To help app vendors formulate cost-effective EDD strategies that minimize the EDD cost while fulfilling the app vendor's EDD time constraint, this paper makes the first attempt to study the EDD problem from the app vendor's perspective. The key contributions of this paper are as follows:

- We formulate and model the EDD problem as a constrained optimization problem (COP) from the app vendor's perspective and prove that it is $\mathcal{NP}$-hard.
- We develop an optimal approach, namely EDD-IP, for finding optimal solutions to EDD problems with the Integer Programming technique.
- We develop an approximation approach named EDD-A for finding approximate solutions to large-scale EDD problems rapidly.
- We conduct extensive experiments on a widely-used real-world dataset to evaluate the proposed approaches against three representative approaches.

The rest of this paper is organized as follows. Section 2 motivates this research with an example. Section 3 formulates the EDD problem and proves its $\mathcal{NP}$-hardness. Section 4 presents and analyzes our optimal approach and approximation approach for solving the EDD problem. Section 5 evaluates the proposed approaches experimentally. Section 6 reviews the related work. Section 7 concludes this paper and points out the future work.

2 Motivating Example

Facebook Horizon[2] is a representative application that can significantly benefit from caching their data on edge servers. Facebook users wearing Oculus headsets can access VR videos and VR games on Facebook Horizon. Caching popular VR videos on edge servers will allow Facebook users covered by those edge servers to access the videos with minimum latency, which is critical because VR users are highly latency-sensitive. It will also reduce the data traffic between the Facebook Horizon server in the cloud and Facebook Horizon users. However, assuming a similar price for data transmission asked by Amazon, cost-ineffective data distribution strategies may cost Facebook Horizon significantly just to distribute VR videos to edge servers. As the number of Facebook Horizon users continues to grow, such extra expense will increase rapidly.

2. https://www.oculus.com/facebookhorizon

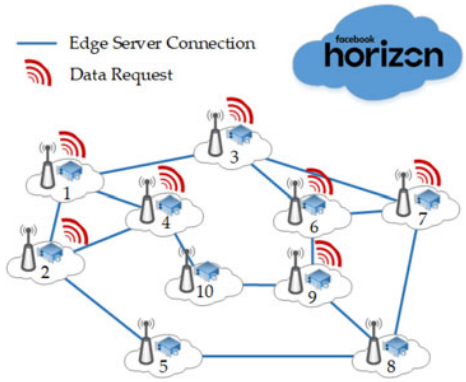


Fig. 1. An example EDD scenario.

Fig. 1 presents an example EDD scenario with 10 edge servers in a specific geographic area. Those edge servers are connected via high-speed links to facilitate data transmissions between them [7], [22]. Let us assume that a Facebook Horizon VR video is to be cached on 7 of those edge servers.[3] There are many possible strategies for distributing the VR video onto those 7 edge servers. A straightforward EDD strategy is to transmit this VR video from the cloud to each individual edge server directly. We refer to such data transmissions as cloud to edge server (C2E) transmissions hereafter. Alternatively, the VR video can first be transmitted from the cloud onto one of the edge servers, which then transmits the VR video onto other edge servers via the high-speed links between them. The data transmissions between edge servers are referred to as edge server to edge server (E2E) transmissions hereafter. A third possible EDD strategy is similar to the second one, but the VR video is first transmitted from the cloud to several of the edge servers instead of one. Given the same amount of data to transmit, E2E transmissions cost less than C2E transmissions because of the much shorter distance between adjacent edge servers and the zero burden caused by E2E transmissions on the internet backbone [23]. Therefore, different EDD strategies incur different costs.

From Facebook's perspective, it is critical to formulate a cost-effective data distribution strategy that minimizes the data distribution cost. While pursing low data distribution cost, the time taken to distribute the VR video onto all the 7 edge servers must also be considered. As discussed above, low latency is one of the major objectives of edge computing [24]. Thus, an EDD strategy must also fulfill Facebook Horizon's time constraint.[4]

3 Problem Formulation

In this section, we first formulate the EDD problem as a constrained optimization problem, then prove the $\mathcal{NP}$-hardness of this problem based on the Steiner Tree problem. The notations used in this paper are summarized in Table 1.

3. When different VR videos are to be cached, their distribution processes are not correlated in terms of transmission cost and transmission delay. Thus, their corresponding EDD strategies are formulated individually.

4. Please note that the time constraint for EDD varies from application to application.

TABLE 1
Summary of Notations

Notation	Description
c	cloud node
d_{limit}	delay limit defined by app vendor
d_v	delay that edge server v obtain data after data arrives the edge server network
$\mathcal{D}_v$	depth of v, equal to $d_v + 1$
$\mathcal{D}_{limit}$	depth limit, equal to $d_{limit} + 1$
E	set of links between edge servers
$e_{u,v}$	edge/link between edge server u and v
G	graph presenting a particular area
$\mathcal{H}$	latency limit defined by app vendor
$\mathcal{R}$	set of destination edge servers
γ	the ratio of data transmission cost of $C2E$ and a 1-hop transmission cost of $E2E$
S	set of binary variables indicating the selection of the initial transit edge servers
s_v	binary variable indicating whether v is an initial transit edge server
$\mathcal{T}$	set of binary variables indicating the data distribution path
T_{ce}	tree with root c with edges $e_{c,v}, \forall v \in \{T_{ms} - c\}$
T_{edd-a}	tree returned by Algorithm 2
T_{ms}	tree returned by Algorithm 1
$\mathcal{T}_{u,v}$	binary variable indicating data transmitted from u to v
V	set of edge servers
v	edge server v
u	edge server u

3.1 Problem Statement

Edge computing is significantly different from cloud computing which facilitates content-centric network and content delivery network. In an edge computing environment, adjacent edge servers deployed at different base stations and access points can communicate with their neighbor edge servers and share their storage resources via high-speed links [7], [22]. Thus, the edge servers in a particular area constitute an *edge server network*, which can be modeled as a graph where a node represents an edge server and an edge represents the link between two edge servers.

In this research, the n edge servers in a particular area are modeled as a graph G. For each edge server v, graph G has a corresponding node. For each pair of linked edge servers (u, v), graph G has a corresponding edge $e_{u,v}$. We use $G(V, E)$ to represent the graph, where V is the set of nodes in G and E is the set of edges in G. *In the remainder of this paper, we will speak inter-changeably of an edge server and its corresponding node in graph G, denoted as $v, \forall v \in V$.* Let $\mathcal{R}$ denote the set of *destination edge servers* in graph G, i.e., the edge servers to which the data are to be transmitted from the cloud.

Example 1. As discussed above, the edge server network in Fig. 1 can be modeled as the graph G presented in Fig. 2, while the destination edge servers are presented as black nodes and others are white nodes. In graph G, there are 10 edge servers with 14 edges in total, where $\mathcal{R} = \{1, 2, 3, 4, 6, 7, 9\}$.

The fees for data transmissions charged by different cloud service providers, e.g., Amazon and Google, are different. Amazon even has different pricing models in its different regions. Similarly, edge infrastructure providers usually have different pricing models for their E2C transmissions and E2E transmissions. Thus, we use a ratio γ to indicate the difference between the C2E transmission cost and the E2E transmission cost generically. For example, $\gamma = 20$ indicates that a C2E transmission costs 20 times as much to transmit a data than an E2E transmission - C2E transmissions are usually more expensive than E2E transmissions as discussed in Section 2. In addition, the data transmission latency between two edge servers is measured by the number of hops between them in graph G. Thus, the C2E transmission cost can be converted to γ times of the 1-hop E2E transmission cost. This way, the optimization objective and the corresponding constraints in the EDD problem can be modeled in a more generic manner. Specific pricing models and network latency models can be easily integrated to calculate the actual EDD cost and EDD time consumption, i.e., the total cost and time of transmitting the data from the cloud to the destination edge servers, respectively.

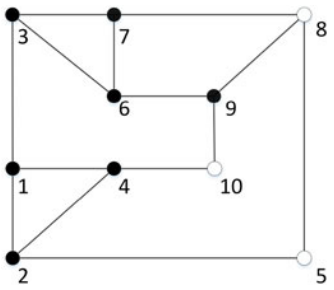


Fig. 2. Graph structure of EDD scenario in Fig. 1.

There are two possible phases of EDD. 1) The $C2E$ transmission, where the data is transmitted from the cloud to one or many of the edge servers in the area, which are referred to as the "initial transit edge servers" hereafter. 2) The $E2E$ transmission, where the data is transmitted from the initial transit edge servers to the destination edge servers via other transit edge servers. Please note that an initial transit edge server is not necessarily a destination edge server. A destination edge server may also be a transit edge server because it may transmit the data further to other destination edge servers. Accordingly, an EDD strategy consists of two parts, a C2E strategy and an E2E strategy. A C2E strategy specifies the initial transit edge servers. It is denoted as a vector $S = < s_1, \ldots, s_n >$, where s_v $(1 \leq v \leq n)$ indicates whether edge server v is selected as an initial transit edge server to receive the data from the cloud directly

$$s_v = \begin{cases} 0 & \text{if } v \text{ is selected as an initial transit edge server} \\ 1 & \text{if } v \text{ is NOT selected as an initial transit edge server} \end{cases}. \tag{1}$$

An E2E strategy is also represented by a vector $\mathcal{T}_{E2E} = < \mathcal{T}_{1,1}, \mathcal{T}_{1,2}, \ldots, \mathcal{T}_{n,n} >$, where $\mathcal{T}_{u,v}(u, v \in V)$ denotes whether the data is transmitted through edge $e_{u,v}$ in G

$$\mathcal{T}_{u,v} = \begin{cases} 1 & \text{if data is transmitted through edge } e_{u,v} \\ 0 & \text{if data is NOT transmitted through edge } e_{u,v} \end{cases}. \tag{2}$$

Since a valid EDD strategy must connect each destination edge server $v \in R$ to an initial edge server in S through $\mathcal{T}_{E2E}$, constraint (3) must be fulfilled

$$isConnected(v, S, \mathcal{T}_{E2E}) = true, \forall v \in R. \tag{3}$$

The details to fulfil constraint (3) will be discussed later in Theorem 3.

As discussed above, the EDD time constraint is determined by the app vendor - i.e., application-specific. As discussed in Section 1, we formulate the EDD problem in a generic manner. The data transmission latency between two edge servers is measured by the number of hops between them in G. Let d_{limit} denote the app vendor's EDD time constraint. Please note that the EDD time constraint here does not include the C2E latency because it is ensured by the edge infrastructure provider and does not impact the formulation of the EDD strategy - it can never be avoided or reduced by an EDD strategy. Thus, each destination edge server v's *E2E latency*, i.e., the data transmission latency between v and its connected initial edge server in S must not exceed this constraint

$$0 \leq d_v \leq d_{limit}, d_v \in Z^+, \forall v \in \mathcal{R}. \tag{4}$$

Example 2. Take Fig. 2 as an example. Let us assume that the app vendor's EDD time constraint is $d_{limit} = 2$. This means that it must not take more than two hops for a destination edge server to receive the data from an initial transit edge server. In Fig. 2, if node 3 is the only edge server selected as the initial transit edge server, one possible E2E strategy is to select edges $\{e_{3,1}, e_{3,6}, e_{3,7}, e_{1,2}, e_{1,4}, e_{6,9}\}$. This means that in $\mathcal{T}_{E2E}$ there is $\mathcal{T}_{3,1} = \mathcal{T}_{3,6} = \mathcal{T}_{3,7} = \mathcal{T}_{1,2} = \mathcal{T}_{1,4} = \mathcal{T}_{6,9} = 1$. Accordingly, we can obtain the 7 destination edge servers' E2E latency: $d_3 = 0$, $d_1 = d_6 = d_7 = 1$, $d_2 = d_4 = d_9 = 2$.

Given an EDD time constraint d_{limit}, the app vendor's optimization objective is to minimize the EDD cost, which consists of the part incurred by the C2E transmission(s) and the E2E transmissions

$$\text{minimize } Cost_{C2E}(S) + Cost_{E2E}(\mathcal{T}_{E2E}), \tag{5}$$

while fulfilling the EDD time constraint (4).

3.2 Problem Hardness

Now we prove that the EDD problem is $\mathcal{NP}$-hard by proving Theorems 1 and 2.

Theorem 1. *The EDD problem is $\mathcal{NP}$.*

Proof. As there are no more than $(|V| + |E| + 2|R|)$ constraints in total, any solution to this EDD problem can be validated in polynomial time by checking whether the solution satisfies the constraints (1), (2), (3) and (4). Thus, the EDD problem is $\mathcal{NP}$. □

Theorem 2. *The EDD problem is $\mathcal{NP}$-hard.*

Proof. To prove the $\mathcal{NP}$-hardness of the EDD problem, we first introduce the classic Rooted Minimum Steiner Tree (RMST) problem. The RMST problem is well-known to be $\mathcal{NP}$-hard [25], [26], and can be defined as follows. Given a graph $G = (V, E)$, a set of destination nodes $\mathcal{N}$ in G, and a root node $root$ of the Steiner tree ST. For each edge $e \in E$, there is a variable $\mathcal{Y}_e$ to indicate whether it is in ST ($\mathcal{Y}_e = 1$) or not ($\mathcal{Y}_e = 0$). Moreover, each edge e has its own weight $\mathcal{W}_e$. Function $path(n, root, ST)$ is used to obtain the possible path from $root$ to the node n through the edges in E. If node n is the root of ST or not in ST, there should not exist a path from root to n, which means $path(n, root, ST) = null$. The formulation is presented below

$$object: \min \sum \mathcal{Y}_e \cdot \mathcal{W}_e \tag{6a}$$

$$s.t.: \ \mathcal{Y}_e \in \{0, 1\} \tag{6b}$$

$$path(n, root, ST) \neq null, \forall n \in \mathcal{N}. \tag{6c}$$

Now we prove that the RMST problem can be reduced to an instance of the EDD problem. The reduction can be done as follows: 1) add the cloud server as a node r into G to obtain a new graph G'; 2) add the edges from node r to every other node in G'; 3) relax the EDD time constraint d_{limit} to $|V|$. Given any instance $RMST(G, root, R, \mathcal{W})$, we can correspondingly construct $EDD(G', cloud, R, Cost_{C2E}, Cost_{E2E})$ with the reduction above in polynomial time where $|G| = |G'|$, while $Cost_{C2E}$ and $Cost_{E2E}$ can be treated as the weights of the edges. By the reduction, the constraint (4) can be relaxed properly. As the constraint (6b) in the RMST problem only considers the edge variables, we can convert constraint (1) to $S_v = \mathcal{T}_{c,v} \in \{0, 1\}$. By combining this with constraint (2), constraint (6b) is fulfilled. Additionally, both constraints (3) and (6c) ensure that all the nodes in R are connected to $root$. Moreover, objective (6a) of RMST is to obtain the minimum total weight of ST, which can be projected to (5) by mapping the $Cost_{C2E}$ and $Cost_{E2E}$ to the weights of edges.

In conclusion, any solution $\mathcal{Y}$ always satisfies the RMST problem if $\mathcal{Y}$ satisfies the reduced EDD problem. Therefore, the EDD problem is reducible from the RMST problem and it is thus $\mathcal{NP}$-hard. □

4 Edge Data Distribution Strategy Formulation

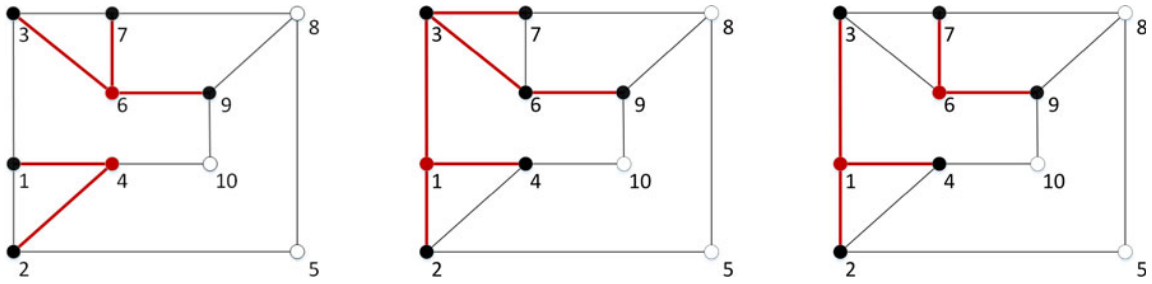
We first present an optimization approach, namely EDD-IP, to exactly solve the EDD problem formulated in the previous section. Then, we introduce an $O(k)$-approximate approach named EDD-A to solve large-scale EDD problems approximately, followed by a theoretical analysis of its performance.

4.1 Optimization Approach

The solution to the EDD problem must minimize the EDD cost while fulfilling the app vendor's EDD time constraint D_{limit}. Thus, the EDD problem can be modeled as a constrained optimization problem.

Following the methodology of the proof of Theorem 3, we use similar techniques to convert the EDD problem to an integer program model that can be solved by integer programming solvers, such as IBM CPLEX Optimizer[5] and

5. https://www.ibm.com/analytics/cplex-optimizer



(a) EDD-IP solution (b) CMST solution (c) EDD-A solution

Fig. 3. EDD solutions.

Gurobi.[6] Accordingly, this optimal approach is named EDD-IP.

First, we add the cloud c into V, then add the edges from c to each edge server $v \in V \setminus c$ in graph G. Then, the C2E strategy $S =< s_1, \ldots, s_n >$ can be formulated by selecting edges in graph G, i.e., $\mathcal{T}_c =< \mathcal{T}_{c,1}, \mathcal{T}_{c,2}, \ldots, \mathcal{T}_{c,n} >$, where $\mathcal{T}_{c,v}$ $(v \in V \setminus c)$ denotes whether the data is transmitted from the cloud c to v through the new edge $e_{c,v}$. Here, we combine the C2E strategy S and the E2E strategy $\mathcal{T}_{E2E}$ as the data distribution strategy $\mathcal{T} = < \mathcal{T}_{c,1}, \ldots, \mathcal{T}_{c,n}, \mathcal{T}_{1,1}, \ldots, \mathcal{T}_{n,n} >$, where variable $\mathcal{T}_{u,v} \in \{0,1\}$ $(u \in \{c\} \cup [1,n], v \in [1,n])$ indicates whether edge $e_{u,v}$ is included in $\mathcal{T}$. Second, we implement the definition of depth in the Steiner tree to represent the order of transmitting the data. Denote $\mathcal{D}_v$ as the depth of edge server v, where $\mathcal{D}_v = d_v + 1$ and $\mathcal{D}_c = 0$. Then, the EDD time constraint d_{limit} can be defined as a depth limit $\mathcal{D}_{limit} = d_{limit} + 1$.

After this, we define the variable for each edge server v

$$I_v = \begin{cases} 0 & \text{if } v \text{ is visited during the EDD process} \\ 1 & \text{if } v \text{ is NOT visited during the EDD process} \end{cases}. \tag{7}$$

Now, the COP model for the EDD problem is formally expressed as follows:

$$\min \gamma \sum_{v \in V \setminus c} \mathcal{T}_{c,v} + \sum \sum\nolimits_{v,u \in V \setminus c} \mathcal{T}_{u,v} \tag{8}$$

$$I_v = 1, \forall v \in R \tag{9}$$

$$\sum_{u \in V} \mathcal{T}_{u,v} = I_v, \forall u, v \in V \setminus c \tag{10}$$

$$\mathcal{T}_{u,v} \leq I_u \cdot I_v, \forall u, v \in V \tag{11}$$

$$I_v, \mathcal{T}_{u,v} \in \{0,1\}, \forall u, v \in V \tag{12}$$

$$\mathcal{D}_c = 0 \tag{13}$$

$$\mathcal{D}_c < \mathcal{D}_v \leq \mathcal{D}_{limit}, \forall v \in V \setminus c \tag{14}$$

$$\mathcal{D}_v - \mathcal{D}_u = 1, \forall u, v \in V, \mathcal{T}_{u,v} = 1, \tag{15}$$

6. http://www.gurobi.com/

where γ is the ratio that generically indicates the ratio of the C2E transmission unit cost over the E2E transmission unit cost, as introduced in Section 3.1.

Example 3. Fig. 3a shows the EDD strategy formulated by EDD-IP with EDD time constraint $d_{limit} = 1$. The C2E strategy specifies that nodes 6 and 4 are selected as the initial transit edge server. They receive the data from the cloud and then transmit it to all other destination edge servers. The EDD strategy selects edges $\{e_{c,4}, e_{c,6}, e_{6,3}, e_{6,7}, e_{6,9}, e_{4,1}, e_{4,2}\}$ for data transmissions. The total cost is $2\gamma + 6$ times of a 1-hop E2E transmission cost.

Theorem 3. *EDD-IP computes an optimal solution to the EDD problem.*

Proof. Let $\mathcal{C}_{C2E}$ and $\mathcal{C}_{E2E}$ denote the unit costs of C2E data transmission and E2E data transmission, respectively. This way, objective (5) can be converted to $\mathcal{C}_{C2E} \cdot \sum_{v \in V \setminus c} \mathcal{T}_{c,v} + \mathcal{C}_{E2E} \cdot \sum \sum_{u,v \in V \setminus c} \mathcal{T}_{u,v}$, with $Cost_{C2E}(S)$ calculated by $\mathcal{C}_{C2E} \cdot \sum_{v \in V \setminus c} \mathcal{T}_{c,v}$ and $Cost_{E2E}(\mathcal{T}_{E2E})$ calculated by $\mathcal{C}_{E2E} \cdot \sum \sum_{u,v \in V \setminus c} \mathcal{T}_{u,v}$. Denote $\gamma = \frac{\mathcal{C}_{C2E}}{\mathcal{C}_{E2E}}$, objective (5) can be presented as $\mathcal{C}_{E2E} \cdot (\gamma \sum_{v \in V \setminus c} \mathcal{T}_{c,v} + \sum \sum_{u,v \in V \setminus c} \mathcal{T}_{u,v})$. Since $\mathcal{C}_{E2E}$ is a constant, objective (5) can be transformed to objective (8) by removing $\mathcal{C}_{E2E}$.

There are two situations in Eq. (10): 1) if v is not visited, any edge pointing to v will not be selected in the C2E strategy; 2) if v is visited, there must be exactly selected one edge that points to v. Constraints (9) and (10) ensure that all the destination edge servers in R are visited during the EDD process. Constraint (11) ensures that, if edge $e_{u,v}$ is selected, both edge servers u and v must be selected. Otherwise, $e_{u,v}$ can never be included into the E2E strategy.

Thus, constraints (9), (10), (11) and (12) collectively ensure that every destination edge server $v \in R$ is connected to the cloud server in G, fulfilling (3) in Section 3.1.

Constraint (13) makes sure that the EDD process always starts from the cloud. In addition, constraint (14) ensures that the EDD time constraint is fulfilled, while constraint (15) guarantee the the depth of u is always one less than that of v, if edge $\{u, v\}$ exists.

Thus, EDD-IP computes an optimal solution to the EDD problem. □

As discussed in Section 3.1, specific cost models and latency models can be easily integrated to calculate the actual EDD costs and EDD time consumption in real-world

EDD scenarios. For example, given a cost function $cost(u, v)$ that represents the transmission cost between u and v, the total EDD cost can be calculated by $\sum_{v \in V \setminus c} \mathcal{T}_{c,v} \cdot cost(c, v) + \sum \sum_{u,v \in V \setminus c} \mathcal{T}_{u,v} \cdot cost(u, v)$. Given a specific latency constraint $\mathcal{L}_{limit}$ and a latency function $\mathcal{L}(u, v)$ that represents the latency between u and v via $e_{u,v}$, constraints (13) (14) and (15) can be replaced by (16), (17) and (18) respectively

$$\mathcal{L}_{\mathcal{T}}(c, c) = 0 \tag{16}$$

$$\mathcal{L}(c, v) \leq \mathcal{L}_{\mathcal{T}}(c, v) \leq \mathcal{L}_{limit}, \forall v \in V \setminus c \tag{17}$$

$$\mathcal{L}_{\mathcal{T}}(c, v) - \mathcal{L}_{\mathcal{T}}(c, u) = \mathcal{L}(u, v), \forall u, v \in V, \mathcal{T}_{u,v} = 1, \tag{18}$$

where $\mathcal{L}_{\mathcal{T}}(c, v)$ represents the total transmission latency between c and v via the path from c to v indicated by $\mathcal{T}$.

4.2 Approximation Algorithm

As proven in Section 3.2, the EDD problem is $\mathcal{NP}$-hard. Finding the optimal solution is intractable in large-scale EDD scenarios. To address this issue, this section presents an approximation approach, named EDD-A, for finding approximate solutions to large-scale EDD problems efficiently. The approximation ratio of EDD-A is $O(k)$, which means that the ratio of the EDD cost incurred by EDD-A and that incurred by the optimal solution is $O(k)$ in the worst case, where k is a constant.

Similar to the techniques used in Section 4.1, EDD-A is an approach designed based on the concept of Steiner tree by adding the cloud server c and the corresponding edges into G. There are two parts in EDD-A: 1) calculating an approximate minimum Steiner tree T_{ms} based on a simple but fast algorithm presented in Algorithm 6 (CMST); 2) splicing and pruning T_{ms} with the latency constraint $\mathcal{H} = \mathcal{D}_{limit}$, based on Algorithm 2.

We first introduce our Connectivity-oriented Minimum Steiner Tree (CMST) algorithm for calculating the approximation to the minimum Steiner tree. CMST is based on the algorithm proposed in [27] which is simple but effective. In our CMST algorithm, it first collects the nodes closest (with the lowest cost) to T_{ms} as a set, then selects the one with the highest connectivity (Lines 3-4 in Algorithm 1).

Algorithm 1. CMST Algorithm

1: **Input:** $G(V, E)$, R, c
2: **Output:** A minimum Steiner Tree T_{ms}
3: initialize a tree T_{ms} based on $G \bigcup \{cloud\}$, only consisting of the node $\{cloud\}$;
4: return T_{ms} if all the destination edge servers in R have been added into T_{ms}, else go to step 3;
5: find a set of edge servers C that are closest to T_{ms}, while $C \bigcap T_{ms} = \varnothing$
6: find edge server v with the highest connectivity, then add v to T_{ms};
7: go to step 2.

Example 4. Take Fig. 2 as an example. By applying CMST, edges $\{e_{c,1}, e_{1,2}, e_{1,3}, e_{1,4}, e_{3,6}, e_{3,7}, e_{6,9}\}$ are added into T_{ms}, as shown in Fig. 3b. The total cost of T_{ms} is $\gamma + 6$. Let us assume that all E2E transmissions must be finished within one hop ($d_{limit} = 1$) to facilitate the following discussion. This means $\mathcal{H} = 2$. Thus, Nodes 6, 7 and 9 violate this limit. Thus, the EDD-A algorithm needs to fix such violations.

Algorithm 2. EDD-A Algorithm

1: **Input:** $G(V, E)$, R, c, $\mathcal{H}$, T_{ms}
2: **Output:** A low-cost Steiner Tree T_{edd-a} within $\mathcal{H}$
3: $T_{edd-a} \leftarrow T_{ms}$, $parents[c] \leftarrow null$
4: For each edge server $v \in G$, set the parent of v in T_{edd-a} as $parents[v]$, the data retrieval latency of edge server v in T_{edd-a} as $d[v]$ where $d[c] \leftarrow 0$, and the cost from v to c in T_{edd-a} as $costs[v]$
5: **for** each edge server v in T_{edd-a} in DFS order **do**
6: **if** $v \notin R$ **or** $d[v] \leq \mathcal{H}$ **then**
7: continue
8: **end if**
9: find the edge server $s \in \{T_{edd-a} - c\}$ that minimizes the cost of path $[c - s - v]$.
10: **if** $\Delta d[s, v] + d[s] \leq \mathcal{H}$ **then**
11: update $parents[]$ and $costs[]$ for edge servers on path $[c - s - v]$, add path $[s - v]$ into T_{edd-a} and update $d[]$
12: **for** each edge server $u \in R$ **do**
13: **if** $costs[u] > costs[v] + cost(v, u)$ **then**
14: $costs[u] \leftarrow costs[v] + cost(v, u)$
15: add path (v, u) into T_{edd-a} and update $d[]$
16: **end if**
17: **end for**
18: **else**
19: $parents[v] \leftarrow c$
20: $costs[v] \leftarrow cost(c, v)$
21: update $d[]$
22: **end if**
23: **for** each child u of $v \in T_{edd-a}$ **do**
24: **if** $costs[u] > costs[v] + cost(v, u)$ **then**
25: $parents[u] \leftarrow v$
26: $costs[u] \leftarrow costs[v] + cost(v, u)$
27: **end if**
28: **if** $costs[v] > costs[u] + cost(u, v)$ **then**
29: $parents[v] \leftarrow u$
30: $costs[v] \leftarrow costs[u] + cost(u, v)$
31: **end if**
32: update $d[]$
33: **end for**
34: **end for**
35: prune unused edges in T_{edd-a} based on $parents[]$
36: **return** T_{edd-a}

Algorithm 2 presents the pseudo code of EDD-A. It takes the minimum Steiner tree T_{ms} returned by Algorithm 1 as input. First, it initializes T_{edd-a} by T_{ms} and sets the parent of cloud server c as null (Line 3). Then, it initializes $parents[]$ for recording the parent of each node in T_{edd-a}, $d[]$ for recording the transmission latency between c and each node in T_{edd-a}, and $costs[]$ for recording the transmission costs between c and each node in T_{edd-a}. After that, the algorithm visits each node v that violates the latency limit $\mathcal{H}$ and finds the minimum-cost path $[c - s - v]$. If path $[c - s - v]$ helps v eliminate the violation, EDD-A adds path $[c - s - v]$ into T_{edd-a} and updates $parents[]$, $d[]$ and $costs[]$ accordingly (Lines 11-17). If the latency limit is still violated, i.e., the sum of latency $\Delta d[s, v]$ and edge server s's latency $d[s]$

exceeds the latency limit $\mathcal{H}$, v will be connected to c directly (Lines 19-21). Next, EDD-A visits each child u of v to update the shortest paths and the parents for both u and v (Lines 23-33). Finally, the algorithm prunes the unused edges in T_{edd-a} (Line 35), and returns T_{edd-a} as the final result.

Example 5. Continuing with Example 2, EDD-A selects node 6 to connect with the cloud node c directly by adding edge $e_{c,6}$. Now, node 9, i.e., the child of node 6 in T_{edd-a}, can obtain the data via 1 hop. After this, EDD-A visits node 7 and finds the shortest path $[c-6-7]$. Thus, EDD-A adds edge $e_{6,7}$ into T_{edd-a}. After running EDD-A, the result is presented in Fig. 3c. The total cost now is $2\gamma+5$, and the EDD time constraint, $\mathcal{H}=1$, is fulfilled. In this case, the EDD strategy formulated by by EDD-A has the same cost as EDD-IP.

In the EDD-A algorithm, the computational overhead of finding the minimum Steiner tree T_{ms} is $O(|R|^2)$. It takes at most $O(|V|^2)$ to read each edge server in T_{edd-a} in DFS order and obtain the shortest path by the Dijkstra algorithm. Thus, the total computational overhead of EDD-A is $O(|R|^2+|V|^3)=O(|V|^3)=O(n^3)$.

In the rest of this section, we prove that EDD-A is an $O(k)$-approximation algorithm based on the following theorems and lemmas, where k is a constant.

Theorem 4. *CMST is a 2-approximation algorithm to calculating the Steiner minimum tree.*

Proof. The original idea of CMST is the same as the algorithm proposed in [27]. The difference between them is the comparison in the connectivity between the nodes that have the same cost. This difference does not impact CMST's approximation ratio. Thus, CMST is a 2-approximation algorithm, the same as the approximation algorithm presented in [27]. □

Let us assume a tree T_{ce} that consists of all the edge servers in T_{ms} and the root c with the edges $[c,v]$, $\forall v \in \{T_{ms}-c\}$. Denote $cost(T_{ce})$, $cost(T_{ms})$ and $cost(T_{edd-a})$ as the total cost of T_{ce}, T_{ms} and T_{edd-a} accordingly.

Theorem 5. *For tree T_{edd-a} produced by EDD-A, the cost of T_{edd-a} is $\frac{2\gamma}{\mathcal{H}}+1$ times of the cost of T_{ms} at most.*

Proof. Let $v_0=c$, and v_i be the ith edge server, where $i \in 1,\dots m$, that introduces additional paths during the DFS traversal. Once the shortest path $[c-v_i]$, which is edge $[c,v_i]$, is added into T_{edd-a}, the cost is exactly $cost_{T_{ce}}(c,v_i)=\gamma$. Moreover, if the path from c to edge server v_i contains edge $[c,v_{i-1}]$ after updating the tree T_{edd-a}, we can obtain $cost_{T_{edd-a}}(c,v_i) \le cost_{T_{ce}}(c,v_{i-1}) + cost_{T_{ms}}(v_{i-1},v_i)$. However, if the direct path from c to v_i is added into T_{edd-a}, it means that $d[v_i]$ exceeds $\mathcal{H}$, and there is $cost_{T_{edd-a}}(c,v_i) \ge \gamma+\mathcal{H}=(1+\frac{\mathcal{H}}{\gamma})\cdot cost_{T_{ce}}(c,v_i)$. Thus, we can obtain the following equation:

$$\begin{aligned}\left(1+\frac{\mathcal{H}}{\gamma}\right)\cdot cost_{T_{ce}}(c,v_i) &\le cost_{T_{edd-a}}(c,v_i)\\ &\le cost_{T_{ce}}(c,v_{i-1})+cost_{T_{ms}}(v_{i-1},v_i).\end{aligned} \tag{19}$$

Summing (19) for all the m edge servers, we can obtain

$$\begin{aligned}&\left(1+\frac{\mathcal{H}}{\gamma}\right)\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i)\\ &\le \sum_{i=1}^{m}(cost_{T_{ce}}(c,v_{i-1})+cost_{T_{ms}}(v_{i-1},v_i,T_{ms})).\end{aligned} \tag{20}$$

Because of $\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i) \ge \sum_{i=1}^{m-1} cost_{T_{ce}}(c,v_i)$, there is

$$\frac{\mathcal{H}}{\gamma}cost(T_{ce}) \le \sum_{i=1}^{m} cost_{T_{ms}}(v_{i-1},v_i), \tag{21}$$

where $cost(T_{ce})=\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i)$.

For each edge server v changing the path in T_{edd-a} but not adding path $[c,v]$, the update cost, $cost_{update}(v)$, must be less than $cost_{T_{ce}}(c,v)$. Thus, the total cost after constructing T_{ms} cannot be more than $\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i)$.

Based on the DFS traversal, each edge is visited twice. Thus, the total cost of $cost_{T_{ms}}(v_{i-1},v_i)$ is no more than twice of $cost(T_{ms})$

$$\sum_{i=1}^{m} cost_{T_{ms}}(v_{i-1},v_i) \le 2\cdot cost(T_{ms}). \tag{22}$$

Thus, the total cost of T_{edd-a} should be bounded by

$$cost(T_{edd-a}) \le cost(T_{ce})+cost(T_{ms}) \le \left(\frac{2\gamma}{\mathcal{H}}+1\right)\cdot cost(T_{ms}). \tag{23}$$

□

Theorem 6. *EDD-A is an $O(k)$-approximation algorithm.*

Proof. As discussed in Theorem 4, the cost of T_{ms} is at most twice the cost of the minimum Steiner tree T. However, the cost of the optimal solution of the EDD problem, OPT, cannot be less than that of the minimum Steiner tree. Thus, we can obtain

$$\frac{cost(T_{edd-a})}{cost(T_{OPT})} \le \frac{cost(T_{edd-a})}{cost(T)} \le 2\left(\frac{2\gamma}{\mathcal{H}}+1\right)=\frac{4\gamma}{\mathcal{H}}+2. \tag{24}$$

From (24), the approximation ratio of EDD-A is $2+\frac{4\gamma}{\mathcal{H}}$. Let $k=2+\frac{4\gamma}{\mathcal{H}}$. As both γ and $\mathcal{H}$ are constant inputs, k is a constant. Thus, EDD-A is an $O(k)$-approximation algorithm where k is a constant. □

Similar to EDD-IP, specific cost and latency models can also be easily integrated into the EDD-A algorithm. Let $\{\gamma_1,\gamma_2,\dots,\gamma_n\}$ denote the C2E costs, $\{\alpha_{1,1},\alpha_{1,2},\dots,\alpha_{n,n}\}$ as the E2E costs, and $\mathcal{L}(u,v)$ as the latency model. Cost functions $cost(c,v)$ and $cost(u,v)$ can be replaced by γ_v and $\alpha_{u,v}$, respectively, in Algorithm 2, and the latency function $\Delta d[s,v]$ can be calculated by $\mathcal{L}(u,v)$. Now we prove that EDD-A is still an $O(k)$-approximation algorithm in real-world EDD scenarios with specific cost and latency models.

Theorem 7. *EDD-A is an $O(k)$-approximation algorithm with specific cost and latency models.*

Proof. We denote $\gamma_{max}=\{\gamma_1,\gamma_2,\dots,\gamma_n\}$ as the maximum C2E cost, $\alpha_{min}=\{\alpha_{1,1},\alpha_{1,2},\dots,\alpha_{n,n}\}$ as the minimum E2E

cost, $\mathcal{L}_{E2E}^{max}$ as the maximum E2E latency and $\mathcal{L}_{C2E}^{min}$ as the minimum C2E latency. In this case, a specific latency limit $\mathcal{L}_{limit}$ is given by the app vendor to replace $\mathcal{H}$. Let $v_0 = c$, and v_i ($i \in \{1, \dots m\}$) be the ith edge server that introduced additional paths during the DFS traversal in Algorithm 2. Once edge $[c, v_i]$, is added into T_{edd-a}, the cost is exactly $cost_{T_{ce}}(c, v_i) = \gamma_i$. If the path from c to edge server v_i contains edge $[c, v_{i-1}]$ after updating tree T_{edd-a}, we can obtain

$$cost_{T_{edd-a}}(c, v_i) \leq cost_{T_{ce}}(c, v_{i-1}) + cost_{T_{edd-a}}(v_{i-1}, v_i). \tag{25}$$

However, if the direct path from c to v_i is in T_{edd-a}, it means $d[i] > \mathcal{L}_{limit}$. Given α_{min} and $\mathcal{L}_{E2E}^{max}$, the ratio of cost over latency for any E2E edge is more than or equals to $\frac{\alpha_{min}}{\mathcal{L}_{E2E}^{max}}$. Denote $\mathcal{H}' = \frac{\alpha_{min}}{\mathcal{L}_{E2E}^{max}}(\mathcal{L}_{limit} - \mathcal{L}_{C2E}^{min})$, we can obtain

$$cost_{T_{edd-a}}(c, v_i) \geq \gamma_i + \mathcal{H}' = \left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) \cdot cost_{T_{ce}}(c, v_i). \tag{26}$$

Combing (26) and (25), (27) stands

$$\begin{aligned} \left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) \cdot cost_{T_{ce}}(c, v_i) \leq \\ cost_{T_{ce}}(c, v_{i-1}) + cost_{T_{ms}}(v_{i-1}, v_i). \end{aligned} \tag{27}$$

Summing (27) for all the m edge servers, we can obtain

$$\begin{aligned} &\sum_{i=1}^{m}\left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) cost_{T_{ce}}(c, v_i) \\ &\leq \sum_{i=1}^{m}(cost_{T_{ce}}(c, v_{i-1}) + cost_{T_{ms}}(v_{i-1}, v_i)). \end{aligned} \tag{28}$$

Since γ_i is always lower than or equals to γ_{max}, we can obtain

$$\left(1 + \frac{\mathcal{H}'}{\gamma_{max}}\right) \sum_{i=1}^{m} cost_{T_{ce}}(c, v_i) \leq \sum_{i=1}^{m}\left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) cost_{T_{ce}}(c, v_i). \tag{29}$$

Because of $\sum_{i=1}^{m} cost_{T_{ce}}(c, v_i) \geq \sum_{i=1}^{m-1} cost_{T_{ce}}(c, v_i)$ and (29), there is

$$\frac{\mathcal{H}'}{\gamma_{max}} cost(T_{ce}) \leq \sum_{i=1}^{m} cost_{T_{ce}}(v_{i-1}, v_i), \tag{30}$$

where $cost(T_{ce}) = \sum_{i=1}^{m} cost_{T_{ce}}(c, v_i)$.

Based on Theorem 4, we can obtain $k = 2 + \frac{4\gamma_{max}}{\mathcal{H}'}$. The corresponding process is omitted here because it is the same as in Theorems 5 and 6. Thus, Theorem 7 holds. □

5 Experimental Evaluation

We have experimentally evaluated the performance of EDD-IP and EDD-A. All experiments were conducted on a Windows-10 machine equipped with Intel Core i7-8665U processor (4 CPUs, 1.90 GHz) and 8 GB RAM.

5.1 Simulation Settings

5.1.1 Approaches in Comparison

In our experiments, we evaluate the performance of EDD-IP and EDD-A against three representative approaches:

- *EDD-IP*: This approach finds the optimal EDD solutions by solving the COP defined in Section 4.1 with IBM's CPLEX CP Optimizer. Specifically, IloConstraint[7] is used to implement the constraints in EDD-IP, including constraint (11) with the product terms of binary variables.
- *EDD-A*: This approach finds near-optimal EDD solutions with Algorithm 2 described in Section 4.2.
- *Minimum-cost Multi-cast Routing (MMR)* [28]: This approach deals with the data routing problem in communication networks. It is also based on Steiner tree and presented as Algorithm 1 in [28].
- *Greedy Connectivity (GC)*: In this approach, we define the connectivity of edge server as the number of edge servers in R that have yet to receive the data. This approach always selects the edge servers with the highest connectivity to receive data from the cloud, which will then transmit the data to other destination edge servers in R, until all the destination edge servers in R can receive the data within the EDD time constraint d_{limit}.
- *Random*: This approach randomly selects edge servers to receive the data from the cloud, which then transmit the data to other destination edge servers in R , one after another, until all the destination edge servers in R receive the data within the EDD time constraint d_{limit}.

5.1.2 Experiment Data

Two sets of experiments are conducted on a widely-used EUA dataset[8][29]. This dataset contains the geographical locations of 1,464 real-world base stations in Melbourne, Australia. As discussed in Section 3.1, γ is dependent on specific edge infrastructure providers. In the experiments, we set $\gamma = 20$. The links between edge servers are randomly generated and to ensure a connected graph, based on the widely-used Erdös Rényi random graph model [30].

5.1.3 Experiment Parameters

To simulate different EDD scenarios, four parameters that impact the performance of EDD-IP and EDD-A are varied in the experiments.

- *The number of edge servers (n) in G*. This parameter impacts the size of graph G.
- *Edge density ($density$)*. We define the edge density with $density = |E|/n$. This parameter impacts the density of graph G.
- *Ratio of destination edge servers ($ratio$)*. This ratio is calculated by $ratio = |R|/n$. It determines the number of destination edge servers in graph G.

7. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcpoptimizer.pdf
8. https://github.com/swinedge/eua-dataset

TABLE 2
Parameter Settings

	n	d	$ratio$	$limit$	T_{limit}	$\|variables\|$ in EDD-IP	$\|constraints\|$ in EDD-IP
Set #1.1	10, 15, ..., 35	1.0	0.6	2	∞	42, 62, ..., 142	67, 100, ..., 232
Set #1.2	20	1.0, 1.2..., 2.0	0.6	2	∞	82, 86, ..., 102	133, 141, ..., 173
Set #1.3	20	1.0	0.2, 0.4 ..., 1.0	2	∞	82	125, 129, ..., 141
Set #1.4	20	1.0	0.6	1, 2, ..., 5	∞	82	133
Set #2.1	200, 300, ..., 700	2.0	0.6	2	10	1002, 1502, ..., 3502	1721, 2581, ..., 6021
Set #2.2	200	2.0, 2.4, ..., 4.0	0.6	2	10	1002, 1082, ..., 1402	1721, 1881, ..., 2521
Set #2.3	200	2.0	0.2, 0.4, ..., 1.0	2	10	1002	1641, 1681, ..., 1801
Set #2.4	200	2.0	0.6	1, 2, ..., 5	10	1002	1721
Set #2.5	200	2.0	0.6	2	2, 4, ..., 10	1002	1721

- *EDD time constraint (limit)*. Measured by the number of hops, this parameter indicates the app vendor's preference for the time taken by the EDD process.

As mentioned in Section 4.2, finding the optimal solutions is intractable in large-scale EDD scenarios. Thus, we limit the maximum running time (in seconds) of EDD-IP in Set #2.

- *Maximum running time (T_{limit})*. EDD-IP will stop and return the optimal solution among all the solutions that have been found within T_{limit}.

Table 2 summarizes the parameter settings. There are two main sets of experiments, Set #1 of small-scale experiments and Set #2 of large-scale experiments. Every time the value of a parameter varies, the experiment is repeated for 100 times and the results are averaged. The last two columns in Table 2 are the number of variables ($3n + dn + 2$), consisting of I $(n + 1)$, $\mathcal{D}$ $(n + 1)$ and $\mathcal{T}$ $(dn + n)$, and the number of constraints $(4n + 2dn + ratio \cdot n + 1)$ based on (9), (10), (11), (13), (14) and (15) in the EDD-IP model presented in Section 4.1.

5.1.4 Performance Metrics

The objective of the EDD problem is to minimize the EDD cost. Thus, this *cost* is a significant metric for evaluating the effectiveness of EDD-IP and EDD-A. In addition, as discussed in Section 1, the applications in the edge computing environment are latency-sensitive. It must not take too much time to find an EDD solution. Thus, the computational overhead is selected to evaluate the efficiency of EDD-IP and EDD-A.

- *EDD cost (cost)*, calculated by (5), the lower the better.
- *Computational overhead (time)*, measured by the time taken by an approach to find the solution, the lower the better.

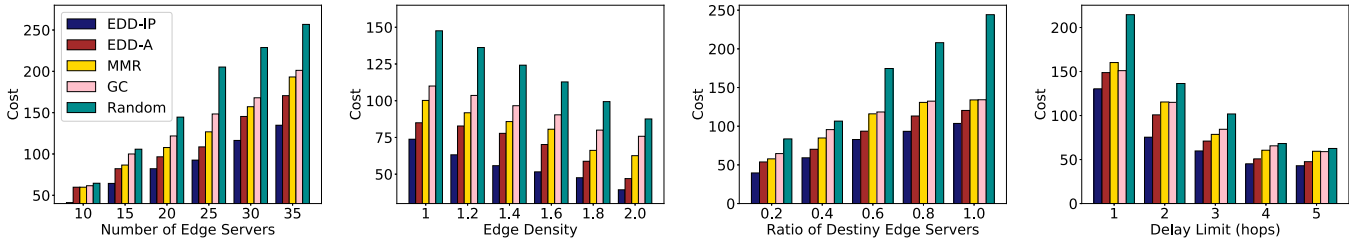
5.2 Experimental Results

5.2.1 Experiment Set #1

Through comparison with MMR, GC and random, Fig. 4 demonstrates the effectiveness of EDD-IP and EDD-A in experiment Set #1 and the impacts of the four parameters. It can be seen that *the EDD strategies formulated by EDD-IP can distribute the data at the lowest cost*, followed by EDD-A. Across the four subsets of experiments, the average advantages of EDD-IP are 18.22 percent over EDD-A, 28.01 percent over MMR, 32.54 percent over GC and 48.34 percent over Random. The average advantages of EDD-A are 11.96 percent over MMR, 17.62 percent over GC and 36.87 percent over Random.

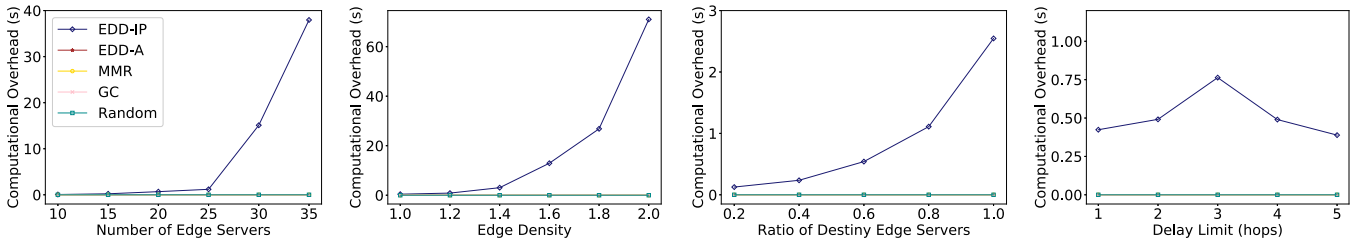
Effectiveness. Fig. 4a shows the effectiveness results of experiment Set #1.1. When the number of edge servers n increases from 10 to 35, the costs of the EDD strategies formulated by all five approaches increase, from 41.2 to 134.8 by 227.18 percent for EDD-IP, from 59.8 to 170.6 by 185.28 percent for EDD-A, from 59.8 to 193.2 by 223.08 percent for MMR, from 61.6 to 201.2 by 226.62 percent by GC, and from 64.6 to 256.8 by 297.52 percent for Random. *Of all the five approaches, EDD-A has the lowest overall increase*.

Fig. 4b shows the results of experiment Set #1.2. Again, *EDD-IP and EDD-A outperform other approaches* with significant margins. The average advantages of EDD-IP are 21.38 percent over EDD-A, 31.97 percent over MMR, 40.43 percent over GC and 53.17 percent over Random, while the numbers for EDD-A are 13.46 percent over MMR, 24.23 percent over GC and 40.44 percent over Random. As the edge density d increases from 1.0 to 2.0, the costs decrease for all the approaches. This is because, with the number of edge servers n fixed, a higher edge density d gives destination edge servers higher chances to receive the data within the EDD time constraint. This reduces the cost incurred by C2E



(a) *cost* vs. n (Set #1.1) (b) *cost* vs. d (Set #1.2) (c) *cost* vs. *ratio* (Set #1.3) (d) *cost* vs. *limit* (Set #1.4)

Fig. 4. Effectiveness comparison in Set #1.



(a) $time$ vs. n (Set #1.1) (b) $time$ vs. $density$ (Set #1.2) (c) $time$ vs. $ratio$ (Set #1.3) (d) $time$ vs. $limit$ (Set #1.4)

Fig. 5. Efficiency comparison in Set #1.

because few destination edge servers have to receive the data directly from the cloud.

Fig. 4c shows the effectiveness results of experiment Set #1.3, where *EDD-IP and EDD-A achieve the best and second best performance*, respectively. The advantage of EDD-IP over EDD-A is 16.09 percent. The advantages of EDD-A over MMR, GC and Random are 13.79, 17.24 and 44.77 percent, respectively. As $ratio$ increases from 0.2 to 1.0, all five approaches need more costs to distribute the data. This is expected because a larger number of destination edge servers will certainly incur higher costs of C2E and/or E2E transmissions.

Fig. 4d shows the results of experiment Set #1.4. *EDD-IP outperforms EDD-A* by 15.56 percent, while EDD-A outperforms MMR, GC and Random by 11.27, 11.79 and 28.20 percent, respectively. As $limit$ increases, the costs of all five approaches decrease. The main reason is that a less stringent EDD time constraint relies less on C2E transmissions which are faster but more expensive than E2E transmissions.

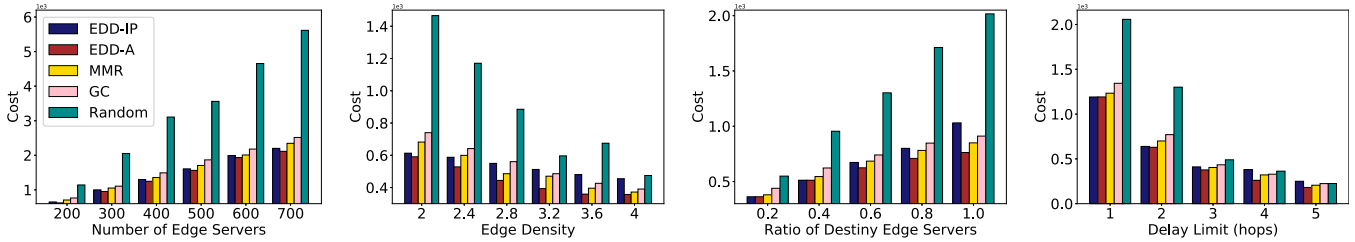
Efficiency. The efficiency results of Set #1 is presented in Fig. 5. As demonstrated, *EDD-IP is much more computationally expensive than all other approaches*. This validates the $\mathcal{NP}$-hardness of the EDD problem - excessive computational overheads are inevitable for finding the optimal solutions to large-scale EDD problems. As demonstrated in Fig. 5a, in largest-scale EDD scenarios with 700 edge servers, EDD-IP takes 39.96 seconds to find the optimal solution in Set #1.1. Moreover, EDD-IP takes up to 71.03 seconds in Set #1.2. When the edge density increases from 1.0 to 2.0, the size of the solution space for EDD-IP to explore becomes larger quickly. Thus, *its computational overhead increases significantly with the increase in edge density*. Similar phenomena are observed in Fig. 5c. Interestingly, the computational overhead of EDD-IP in Fig. 5d increases when the EDD time constraint $limit$ increases from 1 to 3, then decreases after that. With $limit > 3$, EDD-IP can find the optimal solution that requires as few as 1 initial transit edge servers. This eases the EDD-IP's pain in inspecting the possible solutions that require multiple initial transit edge servers. As a result, EDD-IP's computational overhead decreases.

Compared with EDD-IP, *the computational overheads of EDD-A stay at a very low level*, taking less than 8.06 seconds in average to find a solution in the entire Set #1. To further evaluate the performance of EDD-A, we present and discuss the results of Set #2, i.e., large-scale experiments, in Section 5.2.2.

5.2.2 Experiment Set #2

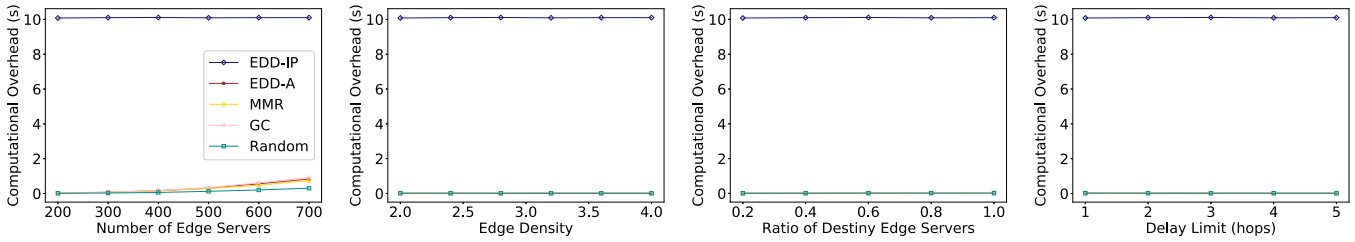
Effectiveness. Fig. 6 demonstrates the EDD costs achieved by EDD-IP, EDD-A, MMR, GC and Random in experiment Sets #2.1 to #2.4 . Overall, their trends in achieving a low cost are similar to Fig. 4 with the same reasons for their performance changes as discussed in Section 5.2.1. Since the maximum running time of EDD-IP is limited in Set #2, its solution is not always the real optimal solution. Thus, EDD-A achieves the best performance in the entire Set #2. In Fig. 6a, the costs increase from 650.3 to 2,200.1 by 238.82 percent for EDD-IP, from 623.4 to 2,115.0 by 239.27 percent for EDD-A, from 706.2 to 2,346.2 by 232.23 percent for MMR, from 764.6 to 2,515.8 by 229.03 percent for GC, and from 1,142.0 to 5,614.6 by 391.65 percent for Random. Fig. 6b shows that the advantages of EDD-A are 29.86 percent over EDD-IP, 11.08 percent over MMR, 17.63 percent over GC and 49.26 percent by Random on average in experiment Set #2.2 where the edge density d increases from 2.0 to 4.0. The performance differences between the five approaches demonstrated in Figs. 6c and 6d are similar to those demonstrated in Figs. 4c and 6d. Thus, it is not discussed in detail here. Fig. 8a depicts that the cost achieved by EDD-IP decreases in Set #2.5 when the running time limit T_{limit} increases. This is because given more running time, EDD-IP can search for more possible solutions to achieve a lower cost.

Efficiency. Fig. 7 depicts the efficiency of each approach in experiment Sets #2.1 to #2.4. In such large-scale experiments,



(a) $cost$ vs. n (Set #2.1) (b) $cost$ vs. $density$ (Set #2.2) (c) $cost$ vs. $ratio$ (Set #2.3) (d) $cost$ vs. $limit$ (Set #2.4)

Fig. 6. Effectiveness comparison in Sets #2.1 - #2.4.



(a) $time$ vs. n (Set #2.1) (b) $time$ vs. $density$ (Set #2.2) (c) $time$ vs. $ratio$ (Set #2.3) (d) $time$ vs. $limit$ (Set #2.4)

Fig. 7. Efficiency comparison in Sets #2.1 - #2.4.

EDD-A, MMR, GC and Random can still find their solutions very fast. In Set #2.1, EDD-A takes a little more time than MMR and Random when the number of edge servers increases. This is the performance price to pay for EDD-A's advantages over these approaches in terms of effectiveness as shown and discussed above. With the significant advantages of EDD-A, especially where there are more edge servers, a higher edge density, a higher ratio of destination edge servers and a lower EDD time constraint, it is worth running EDD-A in real-world applications. As shown in Fig. 8b, the computational overheads of EDD-IP are slightly higher than its running time limit. The reason is that after EDD-IP stops searching for possible solutions when T_{limit} is reached, it still needs some time to find the best solution among those possible solutions.

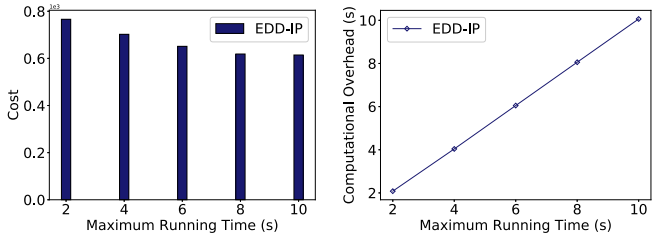
5.2.3 Conclusion

The experimental results demonstrated and discussed above indicate that EDD-IP is suitable for solving EDD problems of reasonable sizes. To solve large-scale EDD problems, EDD-A is more practical for its high efficiency in finding solutions very close to the optimal ones.

5.3 Threats to Validity

5.3.1 Construct Validity

The main threats to the construct validity are threefold: 1) the comparison with MMR, GC and Random may not suffice to comprehensively evaluate EDD-IP and EDD-A; 2) the dataset used in the experiments may not represent all real-world scenarios exactly; 3) the performance of our approaches in real-world EDD scenarios may not be exactly the same as in our experiments. To minimize these threats, we varied setting parameters in both sets of experiments, as summarized in Table 2, to simulate a broad range of various EDD scenarios. This has allowed us to evaluate our approaches more comprehensively. Moreover, we evaluated EDD-IP and EDD-A by not only the comparison with three representative approaches but also the demonstration of how the changes in the setting parameters impacted their performance. In this way, the experimental results can be used as guidelines on the estimation of the performance of our approaches in real-world applications.



(a) $cost$ vs. T_{limit} (Set #2.5) (b) $cost$ vs. T_{limit} (Set #2.5)

Fig. 8. Effectiveness and efficiency of EDD-IP in Set #2.5.

5.3.2 External Validity

The main threat to the external validity of the evaluation is whether EDD-IP and EDD-A can be generalized and applied in other application scenarios in the edge computing environment. To tackle this threat, we modeled the problem and evaluated the performance of EDD-IP and EDD-A in a generic manner - using the number of hops to represent the data transmission latency between edge servers and a ratio γ to indicate the difference between the C2E transmission cost and the E2E transmission cost. In this way, the evaluation results can be interpreted with specific latency and cost models. We also varied parameters to change the size and the complexity of the EDD problem. This way, the representativeness and comprehensiveness of the evaluation are ensured. The above mitigates the threat to external validity.

5.3.3 Conclusion Validity

The lack of statistical tests, e.g., chi-square tests, is the major threat to conclusion validity in our paper. To compensate this threat, we have conducted comprehensive and intensive experiments to cover various scenarios in different size and complexity. In addition, every time a parameter changes, we repeat the experiment for 100 times and calculate the averaged results. This led to a large number of test cases, which tend to result in a small p-value in the chi-square tests and lower the practical significance of the test results [31]. For example, in experiment Set #1, there were a total of 2,200 runs. This number is not even close to the number of observation samples that concern Lin *et al.* in [31]. This way, the threat to the conclusion validity due to the lack of statistical tests might be high but is not significant.

6 Related Work

Edge computing was proposed by Cisco in 2012 as a new computing paradigm. As an extension of cloud computing, edge computing distributes cloud-like computing resources and services to the edge of the cloud [24]. Applications, services and data can now be deployed on edge servers to

offer end-users ultra-low latency. It offers new opportunities and in the meantime raises many new challenges, e.g., edge server placement, edge user allocation, computation offloading, edge application deployment, edge data caching and edge data distribution.

Edge server placement is a fundamental problem in edge computing. In [32], the authors focused on minimizing the cost incurred during edge server deployment. They designed a cost-effective method that employs integer programming to help edge infrastructure providers make decisions on edge server placements. Similarly, Yin *et al.* [33] presented a decision support framework based on flexible placement, namely Tentacle. It aims to minimize the cost of edge infrastructure deployment while maximizing the overall system performance.

The edge user allocation problem in the edge computing environment was first studied in [29]. The authors of [29] modeled this problem as a variable sized vector bin packing problem to maximize the number of allocated app users, while minimizing the cost of hiring edge servers. He *et al.* [3] tackled a similar problem in edge user allocation with the aim to find a near-optimal solution in an efficient manner. They proposed EUAGame, a game-theoretic approach that employs a decentralized algorithm to find the Nash equilibrium of the game as the solution to the problem.

The problem of computation offloading has been intensively studied with consideration of edge servers' energy efficiency, offloading cost and joint service with caching [5], [32], [34], [35]. Xu *et al.* [5] proposed an online algorithm, namely OREO, to efficiently improve offloading performance jointly with service caching. OREO was developed based on Lyapunov optimization to reduce computation offloading latency while keeping energy consumption low. In [36], Wang *et al.* considered the computation offloading problem in wireless cellular networks with mobile edge computing. They modeled this problem as a convex problem and then decomposed it to be solved in a distributed manner.

Application deployment is another problem in the edge computing environment that has attracted increasing attention from researchers. A number of approaches have been proposed to determine optimal deployment strategies with different objectives, such as maximizing the user or request coverage [37], minimizing the deployment cost [38], and maximizing the profit [39]. For example, Wang *et al.* [38] focused on an application migration problem and proposed a Markov decision process based framework for migrating application instances between edge-clouds, with the aim to minimize the average migration cost and the transmission cost over time. Mahmud *et al.* [39] proposed a new pricing model deploying applications on fog instances. They also proposed a user compensation method based on SLA violation. Based on the new pricing model and the user compensation method, an approach was proposed to find optimal application deployment strategies that fulfil resource constraints like processing cores and memory.

In recent years, researchers have started to propose and investigate new ideas and techniques for data caching in the edge computing environment. Cao *et al.*[14] presented an optimal auction mechanism to maximize service provider's revenue based on cache allocation and user valuation reports. They proposed computationally-efficient approaches to apply the auction mechanism based on data retrieval and delivery costs. The authors of [11] proposed a caching system named Cachier for recognition of applications in an MEC environment. Cachier coordinates the loading balance between edge servers and the cloud to minimize the data retrieval latency in a dynamical manner. Breitbach *et al.* proposed a data management system for edge computing environments by decoupling data placement based on task scheduling [40]. The system adjusted the data replica placement cost to achieve the balance between data management overhead and execution delay. In [41], the authors focused on data-intensive IoT workflows in a collaborative edge and cloud computing environment. They also formulated the problem as a 1-0 integer programming model and provided a variant of the intelligent swarm optimization algorithm to solve the problem.

However, existing research has not considered the fact that transmitting the data on cloud and edge computing infrastructure is also a large component in app vendors' cost structure in the edge computing environment. Without considering this component, app vendors will not be able to realistically evaluate the costs of caching their app data on edge servers. To the best of our knowledge, this paper makes the first attempt to solve the edge data distribution problem from the app vendor' perspective in the edge computing environment. Its aim is to minimize the cost of distributing data from the cloud to edge server, with consideration of the costs incurred during C2E and E2E transmissions, while fulfilling the app vendor's time constraint for data distribution.

7 Conclusion

In this paper, we formulated the edge data distribution problem in the edge computing environment as a constrained optimization problem from the app vendor's perspective. We proved that the EDD problem is $\mathcal{NP}$-hard. To solve this problem, we proposed an optimal approach named EDD-IP based on the Integer Programming technique to minimize the cost incurred during data distribution. As the EDD problem is $\mathcal{NP}$-hard, we also provided an approximation approach named EDD-A for finding approximate solutions to large-scale EDD problems efficiently. Extensive experiments were conducted on a widely-used real-world dataset to evaluate the performance of the proposed approaches. The results showed that our approaches significantly outperformed the representative approaches in various EDD scenarios.

This research has established the foundation for the EDD problem and opened up a number of future research directions. In our future work, we will consider the robustness and fault-tolerance of EDD strategies, and more dynamic and heterogeneous aspects of edge servers.

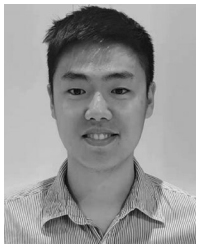
Acknowledgments

This research was funded in part by the Australian Research Council Discovery Projects No. DP180100212, DP200102491, and Laureate Fellowship FL190100035.

References

[1] A. Osseiran *et al.*, "The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions," in *Proc. IEEE 77th Veh. Technol. Conf.*, 2013, pp. 1–5.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[3] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.
[4] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, vol. 12, no. 4, pp. 29–33, 2017.
[5] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.
[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[7] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[8] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[9] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, "Mobile edge computing empowered energy efficient task offloading in 5G," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6398–6409, Jul. 2018.
[10] J. Zhao, W. Gao, Y. Wang, and G. Cao, "Delay-constrained caching in cognitive radio networks," *IEEE Trans. Mobile Comput.*, vol. 15, no. 3, pp. 627–640, Mar. 2016.
[11] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 276–286.
[12] X. Zhang and Q. Zhu, "Hierarchical caching for statistical QoS guaranteed multimedia transmissions over 5G edge computing mobile wireless networks," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 12–20, Jun. 2018.
[13] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5G networks with mobile edge computing," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 80–87, Jun. 2018.
[14] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *Proc. 38th IEEE Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 388–399.
[15] R. Halalai, P. Felber, A.-M. Kermarrec, and F. Taïani, "Agar: A caching system for erasure-coded data," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 23–33.
[16] Y. Wang, B. Veeravalli, and C.-K. Tham, "On data staging algorithms for shared data accesses in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 4, pp. 825–838, Apr. 2013.
[17] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 2004–2017, Sep. 2018.
[18] Y. Liu, M. Dong, K. Ota, and A. Liu, "ActiveTrust: Secure and trustable routing in wireless sensor networks," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 9, pp. 2013–2027, Sep. 2016.
[19] K. Gai, L. Qiu, M. Chen, H. Zhao, and M. Qiu, "SA-EAST: Security-aware efficient data transmission for its in mobile heterogeneous cloud computing," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 2, pp. 1–22, 2017.
[20] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.
[21] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "IoT-based big data storage systems in cloud computing: Perspectives and challenges," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 75–87, Feb. 2017.
[22] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018.
[23] M. Patel *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-Edge Comput. (MEC) Industry Initiative*, pp. 1089–7801, 2014.
[24] M. Yannuzzi *et al.*, "A new era for cities with fog computing," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 54–67, Mar./Apr. 2017.
[25] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Berlin, Germany: Springer, 1972, pp. 85–103.
[26] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, vol. 22, no. 1, pp. 55–89, 1992.
[27] H. Takahashi, "An approximate solution for the steiner problem in graphs," *Math. Japonica*, vol. 6, pp. 573–577, 1990.
[28] G. Xue, "Minimum-cost QoS multicast and unicast routing in communication networks," *IEEE Trans. Commun.*, vol. 51, no. 5, pp. 817–824, May 2003.
[29] P. Lai *et al.*, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2018, pp. 230–245.
[30] P. Erdös and A. Rényi, "On random graphs publ," *Math. Debrecen*, vol. 6, pp. 290–297, 1959.
[31] M. Lin, H. C. Lucas Jr, and G. Shmueli, "Research commentary—too big to fail: Large samples and the p-value problem," *Inf. Syst. Res.*, vol. 24, no. 4, pp. 906–917, 2013.
[32] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency Comput. Pract. Experience*, vol. 29, no. 16, 2017, Art. no. e3975.
[33] H. Yin *et al.*, "Edge provisioning with flexible server placement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1031–1045, Apr. 2017.
[34] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.
[35] S. Jošilo and G. Dán, "A game theoretic analysis of selfish mobile computation offloading," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[36] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.
[37] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *Proc. 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 365–375.
[38] 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," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.
[39] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Profit-aware application placement for integrated fog–cloud computing environments," *J. Parallel Distrib. Comput.*, vol. 135, pp. 177–190, 2020.
[40] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2019, pp. 1–10.
[41] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for IoT workflows in collaborative edge and cloud environments," *Comput. Netw.*, vol. 148, pp. 46–59, 2019.



Xiaoyu Xia received the master's degree from The University of Melbourne, Australia, in 2015. He is currently working toward the PhD degree at Deakin University, Australia. His research interests include edge computing, service computing, and software engineering.

Feifei Chen received the PhD degree from the Swinburne University of Technology, Australia, in 2015. She is currently a lecturer at Deakin University, Australia. Her research interests include software engineering, cloud computing, and green computing.

Qiang He (Member, IEEE) received the first PhD degree from the Swinburne University of Technology, Australia, in 2009 and the second PhD degree in computer science and engineering from the Huazhong University of Science and Technology, China, in 2010. He is a senior lecturer at Swinburne, Australia. His research interests include service computing, software engineering, cloud computing and edge computing. For more information please visit https://sites.google.com/site/heqiang/

John C. Grundy (Senior Member, IEEE) received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently an australian laureate fellow and a professor of software engineering at Monash University, Melbourne, Australia. He is an associate editor of the *IEEE Transactions on Software Engineering*, *Automated Software Engineering Journal*, and *IEEE Software*. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. For more information please visit https://sites.google.com/site/johncgrundy/

Mohamed Abdelrazek is an associate professor of software engineering and IoT at Deakin University, Australia. Before joining Deakin University, Australia, in 2015, he worked as a senior research fellow at the Swinburne University of Technology, Australia and Swinburne-NICTA software innovation lab (SSIL). Before 2010, he was the head of Software Development Department at Microtech. For more information please visit at https://sites.google.com/site/mohamedalmorsy/

Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology, China, in 1994. He is a Cheung Kung scholars chair professor of computer science and engineering at the Huazhong University of Science and Technology (HUST), China. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

5.2 Cost-effective Data Distribution Strategies in Online Edge Computing Scenarios

The approach proposed in **Chapter 5.1** is designed to handle data transmissions in an offline manner without considering temporal data dynamics in real-world edge cache systems - users' demands on data vary at different locations over time, as mentioned in **Chapter 4**. Moreover, every time a piece of data is requested by new edge servers in the system, it has to be transmitted from the remote cloud to the destination edge servers through intermediate edge servers in **Chapter 5.1**. Data transmissions in a real-world edge cache system must be handled in an online manner to overcome the above critical limitations. The key is to source data for destination edge servers from other edge servers in the system instead of from the cloud if it is more cost-effective to do so. In this chapter, we study this edge data distribution problem in an online manner, and propose a Lyapunov-based online approach to minimize the total cost while stabilizing low average transmission latency in the long term.

This chapter is based on a submitted manuscript, entitled: Formulating Cost-Effective Data Distribution Strategies Online for Edge Cache Systems, *IEEE Transactions on Parallel and Distributed Systems*, *Major Revision*.

Formulating Cost-Effective Data Distribution Strategies Online for Edge Cache Systems

Xiaoyu Xia, Feifei Chen, Qiang He, *Senior Member, IEEE*, John Grundy, *Senior Member, IEEE*, Mohamed Abdelrazek, Jun Shen, Athman Bouguettaya, *Fellow, IEEE*, and Hai Jin, *Fellow, IEEE*

Abstract—Edge Computing (EC) enables a new kind of caching system in close geographic proximity to end-users by allowing app vendors to cache popular data on edge servers deployed at base stations. This edge cache system can better support latency-sensitive applications. However, transmitting data from the centralized cloud to the edge servers without proper transmission strategies may cost app vendors dearly based on the pay-as-you-go model. Cost-effective data distribution strategies are of particular importance for applications, whose data to be cached at the edge often changes dynamically over time. In this paper, we study this *Online Edge Data Distribution* (OEDD) problem, aiming to minimize app vendors' total transmission cost, while ensuring low transmission latency in the long term. We first model this problem and prove its $\mathcal{NP}$-hardness. We then propose a novel Latency-Aware Online (LAO) approach, an online algorithm for solving this OEDD problem over time with provable performance guarantees. The evaluation of LAO on a real-world dataset demonstrates that it can help app vendors formulate cost-effective edge data distribution strategies in an online manner.

Index Terms—edge computing, data distribution, edge cache system, optimization, online algorithm.

1 Introduction

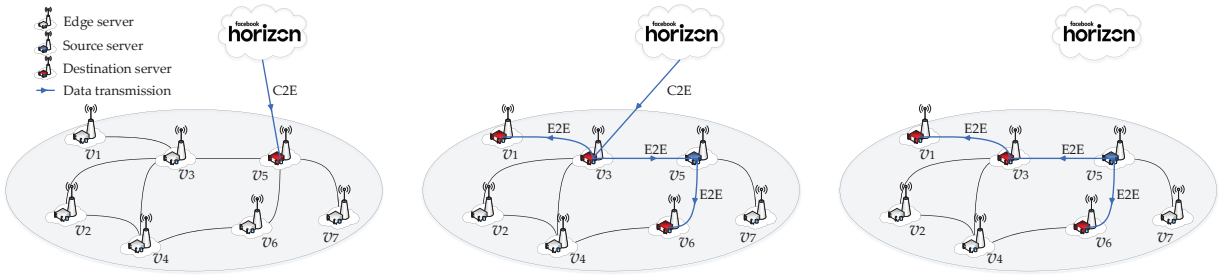
The number of mobile devices has increased exponentially in the last decade, approaching 500 billion by 2030 as predicted by CISCO [1]. These devices generate an enormous load on networks. Considerable network resources are required to transmit such massive data. Conventional caching systems are facilitated by cloud computing, including Content Delivery Networks (CDNs), content-centric networks and information-centric networks. However, those caching systems cannot fulfill the rapidly-increasing need for low latency raised by real-time applications, e.g., AR, VR, online games, etc. Edge Computing (EC) has emerged as the extension of cloud computing to tackle this critical limitation. EC distributes the resources to edge servers deployed at base stations in geographic proximity to nearby end-users [2], [3]. App vendors like Facebook Horizon[1] and Google Stadia[2] can hire storage and computing capacities on edge servers for caching popular data and hosting their applications to serve users within those edge servers' coverage areas with low end-to-end data retrieval latency [4], [5].

Data caching has been intensively investigated to reduce data retrieval latency [6], [7], [8], [9], [10]. A fundamental obstacle to further advances in caching is the often unpredictable high latency caused by the long distance between end-users and the cache [11]. EC addresses this issue by expanding the boundary of data caching to reach geographical proximity to end-users [12], [13]. As edge servers the entry point for an increasing number of devices, there has been an equivalent increase in traffic going through edge servers. Therefore, caching data on edge servers can considerably reduce users' data retrieval latency as there is no need to retrieve data from remote cloud servers [14]. Additionally, this could have the effect of reducing the volume of data transmitted from the cloud to users and decreasing the corresponding *transmission costs* [4]. The edge servers deployed in an area constitute an *edge cache system*. New challenges raised by edge cache systems are starting to attract researchers' attention [13], [15], [16], [17], [18].

Existing research on edge cache systems has focused on caching data across edge servers for different optimization objectives, e.g., maximizing hit ratio [9], [13], minimizing caching cost [4], [16] or minimizing retrieval latency [15], [17]. The fact that transmitting data from the cloud to edge servers may incur excessive costs has been completely ignored. For example, 1 GB data transferred out of data centers to the internet charges up to US$0.11 under Amazon Web Services price model[3]. It is an important consideration for app vendors when leveraging edge cache systems. In addition, taking too long to transmit data onto edge servers impacts users' data retrieval latency and causes user aban-

- *X. Xia, F. Chen and M. Abdelrazek are with School of Information Technology, Deakin University, Geelong, Victoria, Australia. E-mail: xiaoyu.xia@deakin.edu.au; feifei.chen@deakin.edu.au; mohamed.abdelrazek@deakin.edu.au.*
- *Q. He is with School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria, Australia. E-mail: qhe@swin.edu.au.*
- *J. Grundy is with Faculty of Information Technology, Monash University, Melbourne, Victoria, Australia. E-mail: john.grundy@monash.edu.*
- *J. Shen is with School of Computing and Information Technology, University of Wollongong, Wollongong, New South Wales, Australia. E-mail: jshen@uow.edu.au.*
- *A. Bouguettaya is with the School of Computer Science, University of Sydney, Australia. Email: athman.bouguettaya@sydney.edu.au.*
- *H. Jin is with Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, China. Email: hjin@hust.edu.cn.*

1. https://www.oculus.com/facebook-horizon/
2. https://stadia.google.com/
3. https://aws.amazon.com/s3/pricing/



(a) EDD for v_5 in time slot t_0 (b) EDD for v_1, v_3 and v_6 in time slot t_1 (c) OEDD for v_1, v_3 and v_6 in time slot t_1

Fig. 1: Example EDD and OEDD processes. Note: This example presents the distribution of one VR video over time. The processes for distributing different VR videos are not correlated. Thus, their strategies are individually formulated.

donment [19]. Xia et al. studied this *edge data distribution* (EDD) problem [20], aiming to cost-effectively transmit data from the cloud to *destination edge servers*, i.e., the edge servers to cache that data in the edge cache system.

Example. Consider the example of Facebook Horizon (FH), an VR platform. FH application can significantly benefit from caching data like popular[4] VR videos (and the corresponding HTML, CSS, JS and image files) on edge servers [20]. FH users in the edge cache system can retrieve popular VR videos from edge servers with low data retrieval latency. This can also significantly reduce the data transmission cost incurred by the data traffic between FH's cloud server and FH users. An edge cache system is shown in Fig. 1(a), involving a set of edge servers $\{v_1, ..., v_7\}$ covering different geographic areas. The edge servers in the edge cache system constitute a edge server network [20]. The edge server network overcomes the single-point failure problem in the edge-cloud architecture[5] [24], where edge servers can communicate with others through a centralized macro base station. It also avoids unpredictable network latency when edge servers can only communicate via the backhaul network [20], [25]. In time slot t_0, a VR video d is requested by many of the FH users within edge server v_5's coverage areas. As illustrated by Fig. 1(a), d is transmitted from the remote FH cloud server to be cached v_5 via a C2E (cloud to edge server) transmission. Future FH users covered by v_5 can retrieve d with low latency.

However, the approach proposed in [20] is designed to handle data transmissions in an offline manner without considering temporal data dynamics in real-world edge cache systems - users' demands on data vary at different locations over time [26]. Every time a piece of data is requested by new edge servers in the system, it has to be transmitted from the remote cloud to the destination edge servers through intermediate edge servers. Take Fig. 1(b) for example, where edge servers v_1, v_3 and v_6 need to cache VR video d after v_5 in time slot t_1. As demonstrated, the approach proposed in [20] has to transmit d from the cloud to v_3 first via a C2E transmission. Then, v_3 will transmits d to v_1, and v_6 through v_5 via E2E (edge server to edge server) transmissions. As d goes viral over time, d is requested by many other edge servers in the system, e.g., v_2, v_4 and v_7. The inevitable and expensive C2E transmissions can easily incur excessive transmission costs. In the meantime, the E2E transmissions incur *extra* data retrieval latency for users served by v_1 and v_6 compared to those served by v_3 (and those served by v_5 in Fig. 1(a)). This undermines EC's pursuit of low data retrieval latency for users.

Data transmissions in a real-world edge cache system must be handled in an online manner to overcome the above critical limitations. The key is to source data for destination edge servers from other edge servers in the system instead of from the cloud if it is more cost-effective to do so. As illustrated by Fig. 1(c), d can be transmitted from v_5 to v_1, v_3 and v_6. Compared with the strategy presented in Fig. 1(b), this strategy is more cost-effective, especially in the long term, because it greatly reduces the total transmission cost and transmission latency by avoiding slow and expensive C2E transmissions when possible. However, sourcing data from an edge server far away from the destination server(s) may take excessive time and incurs high *transmission latency* which are ignored in [20]. Thus, from the app vendor's perspective, to strike a proper balance, *the cost-effective online EDD (OEDD) must minimize the total transmission cost while stabilizing low average transmission latency in the long term*.

This paper proposes LAO, a **L**atency-**A**ware **O**nline approach, for formulating cost-effective OEDD strategies over time for app vendors. To the best of our knowledge, it is the first attempt at the OEDD problem. The key contributions of this work include:

- We formulate the OEDD problem and prove its $\mathcal{NP}$-hardness.
- We propose an innovative online approach, LAO, for solving the OEDD problem based on the Lyapunov optimization theory, and theoretically prove its performance bounds.
- We evaluate LAO with extensive experiments conducted on a widely-used real-world data set.

4. Data popularity prediction has been investigated extensively in the past two decades. It focuses on cache hit rate, which is orthogonal to how edge cache systems reduce cache hit distance [9], [21], [22], [23]. Thus, the data to be distributed by LAO is assumed to be popular in this study.

5. Edge servers communicating with each other based on the edge-cloud architecture can be modeled as a fully-connected edge server network. Thus, the models and the algorithms proposed in this paper are also suitable for solving OEDD problems based on the edge-cloud architecture.

The rest of this paper is organized as follows. In Section 2, we present system models, formulates the OEDD problem and proves its $\mathcal{NP}$-completeness. Section 3 presents LAO in detail and analyzes its performance theoretically. Section 4 evaluates LAO experimentally against state-of-the-art algorithms. We review the related work Section 5, and then concludes this paper in Section 6.

2 System Model

We introduce the edge cache system architecture first in this section. Then we formally define the transmission cost. After that, we formulate the OEDD problem and prove that this OEDD problem is $\mathcal{NP}$-hard.

2.1 System Architecture

The edge server network in an edge cache system can be modeled as a graph $G(V, E)$, where V is the set of nodes and E is the set of edges in G. Graph G has a corresponding node for each edge server and the cloud server. The link between FH's cloud server and each of the edge servers in the system is also modeled as an edge. With the cloud server modeled as node c, we use $G(V, c, E)$ to represent the entire edge cache system[6].

As discussed in Section 1, a data d can be transmitted to a destination edge server from either the cloud server or another edge server in the system. Denote $\mathcal{S}^t$ as the set of *source servers* in time slot t, including the cloud server c and the edge servers that has d in cache, e.g., v_5 in Fig. 1(b).

Let $\mathcal{R}^t$ denote the set of destination edge servers in time slot t, i.e., the edge servers to which d needs to be transmitted. Let vector $\omega^t =< \omega_0^t, ..., \omega_n^t >$, where $\omega_0^t = \omega_c^t$ and $\omega_v^t \in \{0, 1\}$ $(0 \leq v \leq n)$, indicate whether node v is a *transmission node* in time slot t, i.e., whether v is involved in the transmission of d as the source node, the intermediate node or the destination node. Similarly, let vector $\gamma^t =< \gamma_{0,0}^t, \gamma_{0,1}^t, ..., \gamma_{n,n}^t >$ $(\gamma_{u,v}^t \in \{0, 1\})$ denote whether transmitting d via edge $e_{u,v}$ in time slot t. Notice that $\gamma_{u,v} \equiv 0$ if $u = v$ or $e_{u,v} \notin E$. For each node $v \in V \setminus \mathcal{S}^t$, d is transmitted through edge server v if d is transmitted via an edge connecting another edge server and v:

$$\sum_{u \in V} \gamma_{u,v}^t \geq \omega_v^t, \forall v \in V \setminus \mathcal{S}^t \tag{1}$$

In addition, if d is transmitted via $e_{u,v}$ in time slot t, it will be transmitted through u and v in time slot t:

$$\gamma_{u,v}^t = \omega_v^t \cdot \omega_v^t, \forall u, v \in V \tag{2}$$

2.2 Transmission Cost

The prices for C2E and E2E transmissions are region-specific and vary in edge cache systems built by different edge infrastructure providers. In this paper, we measure transmission costs in a generic manner, similar to [20]. In this way, actual C2E and E2E transmission costs can be calculated based on specific pricing models in real-world scenarios, e.g., Amazon or Google's. Let $cost_{u,v}$ denote the transmission cost incurred by transmitting a piece of data from edge server u to v and $cost_{c,v}$ as the cost of transmitting the data from the remote cloud server c to v. Denoted by $cost^t$, the transmission cost incurred by the EDD strategy γ^t in time slot t is:

$$cost^t = \sum_{v \in V} \gamma_{c,v}^t \cdot cost_{c,v} + \sum_{u \in V} \sum_{v \in V} \gamma_{u,v}^t \cdot cost_{u,v} \tag{3}$$

Please note that the cache spaces on edge servers are constrained and expensive due to their limited sizes [27], [28], [29], [30]. App vendors need to compete for the storage spaces in the edge cache system. A common practice is to reserve cache spaces in advance rather than on-demand [31]. Thus, the expenses of storing data in the edge cache system are fixed and not included in (3).

2.3 Transmission Latency

Let $\bar{l}$ denote the time-averaged maximum transmission latency expected by the app vendor, and $l_v^{\gamma^t}$ denote the time taken to transmit the data to edge server v according to EDD strategy γ^t. As discussed in Section 1, cost-effective OEDD must stabilize low transmission latency in the long term. This is achieved by stabilizing the average time taken

TABLE 1: Summary of Notations

Notation	Description
c	remote cloud
$cost^t$	transmission cost incurred by γ^t
$cost_{u,v}$	cost of transmission through edge $e_{u,v}$
d	data to be cached
E	set of links/edges existing in G
$e_{u,v}$	edge from node u to v
G	graph presenting a particular area
$\bar{l}$	time-averaged maximum transmission latency
$l_{u,v}^t$	time taken to transmit the data via edge $e_{u,v}$ in time slot t
$l_v^{\gamma^t}$	time taken to transmit the data to v according to γ^t
$\mathcal{R}^t$	set of destination edge servers in time slot t
$\mathcal{S}^t$	set of source servers in time slot t
t	time slot t
T	number of time slots
u, v	nodes u, v in graph G
V	set of edge servers
γ	EDD strategy
γ^t	set of binary variables indicating whether data is transmitted through any edge $e_{u,v} \in E$ in time slot t
$\gamma_{u,v}^t$	binary variable indicating whether data is transmitted through $e_{u,v}$ in time slot t
ω^t	set of binary variables indicating whether data transmission strategy visits node $v \in V$ in time slot t
ω_v^t	binary variable indicating whether data transmission strategy visits node v in time slot t

6. In this paper, we speak interchangeably of a server and its corresponding node in G, e.g., source nodes and source servers.

to transmit data to each destination server below $\bar{l}$ over T time slots:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\frac{\sum_{v\in\mathcal{R}^t} l_v^{\gamma^t}}{|\mathcal{R}^t|}\leq \bar{l} \tag{4}$$

In particular, data d does not need to be transmitted to any of the source servers because d is already in their cache. Accordingly, there is:

$$l_v^{\gamma^t}=0, \forall v\in\mathcal{S}^t \tag{5}$$

If d goes through edge $e_{u,v}$ in G according to γ^t, the times taken to transmit d to u and v fulfil:

$$l_u^{\gamma^t}-l_v^{\gamma^t}=l_{u,v}^t, \forall u,v\in V, \text{if } \gamma_{u,v}^t=1 \tag{6}$$

2.4 Problem Formulation and Hardness

As discussed in Section 1, the app vendor wants to minimize the total transmission cost over time. Thus, in a time slot t, d can go through any nodes in G only once. Accordingly, (1) can be refined as follows:

$$\sum_{u\in V}\gamma_{u,v}^t=\omega_v^t, \forall v\in V\setminus\mathcal{S}^t \tag{7}$$

For each destination edge server $v\in\mathcal{R}^t$, it must be included in γ^t, i.e., the EDD strategy in time slot t:

$$\omega_v^t=1, \forall v\in\mathcal{R}^t \tag{8}$$

In the edge cache system, users' data requests arrive and leave randomly [32]. From the app vendor's perspective, the edge cache system's long-term performance usually is superior to its short-term performance. To minimize the total transmission cost over time, we formulate the OEDD problem as follows:

$$\begin{aligned}\mathcal{P}_1:\quad & \min \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1} cost^t\\ s.t.\quad & \gamma_{u,v}^t,\omega_v^t\in\{0,1\}, \forall u,v\in V\\ & (2),(4),(5),(6),(7),(8)\end{aligned}$$

Now, we demonstrate that the t-OEDD problem, i.e., the subproblem of $\mathcal{P}_1$ in an individual time slot t without (4), is $\mathcal{NP}$-hard.

Theorem 1. The t-OEDD problem is $\mathcal{NP}$-hard.

Proof To do this proof, we first introduce a known $\mathcal{NP}$-hard problem, the Rooted Minimum Steiner Tree (RMST) problem: given a graph $G(V,E,R,W,r)$ where V is the set of nodes, E is the set of edges, R is the set of destination nodes $R\in V$, and W is the weights of edges $\forall e\in E$, the RMST problem is to find a Steiner tree from G that starts from node r with minimum weights. Now we present how to reduce the t-OEDD problem to the RMST problem. We first include a virtual node r' into graph G. Then, we add an edge to connect this virtual node to each of the source nodes in G. In the t-OEDD problem, each edge $e_{u,v}$ has a specific transmission cost $cost_{u,v}$, which can be treated as the weight of the corresponding edge in the RMST problem. In this way, the t-OEDD problem is converted to: given a graph G and a starting point r', find the Steiner tree with the minimum total weight (the minimum transmission cost). A solution that fulfills this reduced t-OEDD problem also fulfills the RMST problem. This indicates that the RMST problem can be reduced to the t-OEDD problem and the t-OEDD problem is thus $\mathcal{NP}$-hard. □

The t-OEDD problem is a special case of the OEDD problem where $T=1$. Thus, the OEDD problem over T is also $\mathcal{NP}$-hard.

3 Online Algorithm Design And Analysis

Lyapunov optimization theory has been widely applied in many domains by modeling dynamic systems as queuing systems to stabilize queues like task queues and request queues [33]. LAO leverages Lyapunov optimization theory in an innovative way to stabilize transmission latency over time. In this section, we present LAO in detail and analyze its performance theoretically.

3.1 Latency-Aware Online Algorithm Design

In $\mathcal{P}_1$, constraint (4) aims to stabilize the time-averaged maximum transmission latency. However, it requires complete information about destination edge servers in all time slots, which cannot be obtained in advance in real-world edge cache systems. To tackle this challenge, we introduce the *accumulated latency*.

Definition 1 (Accumulated Latency). Denoted by σ_{t+1} ($t\in[0,T-1]$), the accumulated latency is the overdue delay accumulated in an edge cache system over t time slots. It can be calculated by:

$$\sigma_{t+1}=\left\lfloor\sigma_t+\sum_{v\in\mathcal{R}^t}(l_v^{\gamma^t}-\bar{l})\right\rfloor_+ \tag{9}$$

where $\sigma_0=0$, i.e., the initial value of the accumulated latency before the first time slot is 0. The accumulated latency increases when the average transmission latency exceeds $\bar{l}$, i.e., the expected time-averaged maximum transmission latency. According to Definition 1, the long-term transmission latency constraint (4) can be converted to a new constraint:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\sigma_t]\leq 0 \tag{10}$$

Based on (9), we define $\mathcal{L}(\sigma(t))\triangleq\frac{1}{2}\sigma^2(t)$ to measure the accumulated latency $\sigma(t)$. Then, we define $\Delta(\sigma(t))$ below to introduce constraint (10) as part of the optimization objective to strike the balance between the time-averaged transmission latency and total transmission cost in the long term:

$$\begin{aligned}\Delta(\sigma_t)&=\mathbb{E}[\mathcal{L}(\sigma_{t+1})-\mathcal{L}(\sigma_t)|\sigma_t]=\frac{1}{2}\mathbb{E}[\sigma_{t+1}^2-\sigma_t^2|\sigma_t]\\&=\frac{1}{2}\mathbb{E}[(\sigma_t+\sum_{v\in\mathcal{R}^t}(l_v^{\gamma^t}-\bar{l}))^2-\sigma_t^2|\sigma_t]\\&=\mathbb{E}[\sigma_t\sum_{v\in\mathcal{R}^t}(l_v^{\gamma^t}-\bar{l})|\sigma_t]+\frac{1}{2}\mathbb{E}[(\sum_{v\in\mathcal{R}^t}(l_v^{\gamma^t}-\bar{l}))^2|\sigma_t]\end{aligned} \tag{11}$$

Next, we combine $\Delta(\sigma_t)$ with the optimization objective of $\mathcal{P}_1$:

$$\zeta\cdot\mathbb{E}[cost^t|\sigma_t]+\Delta(\sigma_t) \tag{12}$$

where ζ is the app-specific parameter representing the urgency to maintain the time-averaged transmission latency by lowering current transmission latency when (10) was violated in the previous time slot. In general, a lower ζ will accelerate the stabilization at a potentially higher transmission cost. Its impact on LAO will be experimentally evaluated in Section 4.

According to (11), the calculation of $\Delta(\sigma_t)$ in time slot t requires $\mathcal{L}(\sigma_{t+1})$, which is not available in time slot t. To address this issue, we now try to find the upper bound on (12) that can be calculated based on information available in time slot t. Because $l_v^{\gamma^t}$ is not lower than 0, $\sum_{v\in\mathcal{R}^t}(l_v^{\gamma^t}-\bar{l})^2$ is not higher than $\sum_{v\in\mathcal{R}^t}\bar{l}^2$. Let us define a constant $\Theta = \frac{1}{2}\sum_{v\in\mathcal{R}^t}\bar{l}^2$. Based on (11), the upper bound on $\Delta(\sigma_t)$ fulfils:

$$\Delta(\sigma_t) \leq \sigma_t \cdot \mathbb{E}[\sum_{v\in\mathcal{R}^t}(l_v^{\gamma^t}-\bar{l})|\sigma_t] + \Theta \tag{13}$$

Based on (13), the upper bound on (12) can be found:

$$\begin{aligned}&\zeta\cdot\mathbb{E}[cost^t|\sigma_t]+\Delta(\sigma_t)\leq\\&\zeta\cdot\mathbb{E}[cost^t|\sigma_t]-\sigma_t\cdot\mathbb{E}[\sum_{v\in\mathcal{R}^t}(\bar{l}-l_v^{\gamma^t})|\sigma_t]+\Theta\end{aligned} \tag{14}$$

It can be calculated based on the information available in time slot t. Now, $\mathcal{P}_1$ can be approximated by finding the solution to $\mathcal{P}_2$ below that minimizes the upper bound on (12) according to (14) in each time slot over T:

$$\mathcal{P}_2: \quad \min \quad \zeta\cdot cost^t - \sigma_t\cdot\sum_{v\in\mathcal{R}^t}(\bar{l}-l_v^{\gamma^t})+\Theta \tag{15}$$

$$s.t.: (2), (5), (6), (8), (7)$$

Algorithm 1 **L**atency-**A**ware **O**nline approach (LAO)

1: **Input:** $G(V,c,E), l_{limit}, \bar{l}$, T, $\mathcal{S}=\{\mathcal{S}^0,...,\mathcal{S}^{T-1}\}$
2: **Output:** data distribution strategy $\gamma=\{\gamma^0,...,\gamma^{T-1}\}$
3: $\sigma_0=0$, $t=0$, $\gamma\leftarrow\varnothing$
4: **repeat**
5: Observe the destination edge servers $\mathcal{R}^t$, create a new set $\mathcal{R}'^t=\{v|v\in\mathcal{R}^t\bigcap\neg\mathcal{S}^t\}$ and set $l_v^{\gamma^t}=0,\forall v\in\mathcal{R}^t\bigcap\mathcal{S}^t$
6: Use $\mathcal{R}'^t$ to find the solution γ^t with the minimal value of (15)
7: Calculate $l_v^{\gamma^t}$ for each edge server $v\in R^t$, and update accumulated latency σ_t based on (9)
8: $\gamma=\gamma\bigcup\gamma^t$
9: $t=t+1$
10: **until** $t=T$
11: **return** γ

Algorithm 1 presents the pseudocode of LAO. It begins with initializing the EDD strategies and the accumulated latency (Lines 3). LAO iterates through all the possible EDD strategies fulfilling the constraints in $\mathcal{P}_2$ and chooses the one producing the lowest objective value according to (15) (Lines 5-6). After that, LAO calculates the accumulated latency (Line 7). Please note that Algorithm 1 will iterate over time when used in practice.

3.2 Theoretical Performance Analysis

In this section, we theoretically analyze LAO's performance in minimizing the total transmission cost and stabilizing the time-averaged transmission latency with Theorem 2 and Theorem 3, respectively.

Theorem 2. The total transmission cost of LAO over T time slots is bounded by $O(\frac{1}{\zeta})$ compared with the optimal solution.

Proof Let $\gamma^*=\{\gamma^{*0},...,\gamma^{*T-1}\}$ denote the optimal solution to $\mathcal{P}_1$, which incurs transmission cost $cost^{*t}$. LAO finds the solution to $\mathcal{P}_2$ from its solution space which contains the solution space of $\mathcal{P}_1$. Now, we deduce the upper bound on (12) with respect to $cost^{*t}$:

$$\begin{aligned}&\zeta\cdot\mathbb{E}[cost^t|\sigma_t]+\Delta(\sigma_t)\\&\overset{(14)}{\leq}\zeta\cdot\mathbb{E}[cost^t|\sigma_t]-\mathbb{E}[\sigma_t\cdot\sum_{v\in\mathcal{R}^t}(\bar{l}-l_v^{\gamma^t})|\sigma_t]+\Theta\\&\leq\zeta\cdot\mathbb{E}[cost^{*t}|\sigma_t]-\mathbb{E}[\sigma_t\cdot\sum_{v\in\mathcal{R}^t}(\bar{l}-l_v^{\gamma^{*t}})|\sigma_t]+\Theta\\&\overset{(\dagger)}{\leq}\zeta\cdot\mathbb{E}[cost^{*t}|\sigma_t]-\sigma_t\cdot\sum_{v\in\mathcal{R}^t}(\bar{l}-l_v^{\gamma^{*t}})+\Theta\\&\overset{(\ddagger)}{\leq}\zeta\cdot\mathbb{E}[cost^{*t}|\sigma_t]+\Theta\end{aligned} \tag{16}$$

Note that the optimal solution γ^* is independent of the accumulated latency. Thus, inequality $(\dagger)$ in (16) stands. Inequality $(\ddagger)$ holds because constraint (4) must be fulfilled by the optimal solution γ^* to problem $\mathcal{P}_1$ while $\sigma_t\geq 0$ based on (9). By summing (16) across $t\in[0,T-1]$, we can obtain:

$$\begin{aligned}&\lim_{T\to\infty}\frac{\zeta}{T}\cdot\sum_{t=0}^{T-1}\mathbb{E}[cost^t|\sigma_t]\\&\leq\lim_{T\to\infty}\frac{\zeta}{T}\cdot\sum_{t=0}^{T-1}\mathbb{E}[cost^{*t}|\sigma_t]+\Theta-\frac{\mathbb{E}[\mathcal{L}(\sigma_T)-\mathcal{L}(0)]}{T}\\&\leq\lim_{T\to\infty}\frac{\zeta}{T}\cdot\sum_{t=0}^{T-1}\mathbb{E}[cost^{*t}|\sigma_t]+\Theta\end{aligned} \tag{17}$$

where $\mathcal{L}(\sigma_T)-\mathcal{L}(0)$ is always non-negative according to definition of $\mathcal{L}(\sigma_t)$.

Thus, the margin between the solution γ found by LAO and the optimal solution γ^* is bounded by $\frac{\Theta}{\zeta}$:

$$\lim_{T\to\infty}\frac{1}{T}\cdot\sum_{t=0}^{T-1}\mathbb{E}[cost^t-cost^{*t}|\sigma_t]\leq\frac{\Theta}{\zeta} \tag{18}$$

As mentioned in Section 3.1, Θ is a constant of $\frac{1}{2}\sum_{v\in\mathcal{R}^t}\bar{l}^2$. Thus, the margin between γ and γ^* is $O(\frac{1}{\zeta})$. □

Theorem 3. The time-averaged accumulated latency achieved by LAO is bounded by $O(\zeta)$.

Proof According to (9) and (10), let us assume a positive value $\mathcal{X}$ and the existence of a γ'^t that fulfill:

$$\mathbb{E}[\sum_{v\in\mathcal{R}^t}(l_v^{\gamma'^t}-\bar{l})|\sigma_t]\leq-\mathcal{X} \tag{19}$$

Let $cost_{min}$ and $cost_{max}$ denote the lowest and highest transmission costs achieved by the all the possible solutions

to $\mathcal{P}_1$, respectively. Introducing $cost_{min}$ and $cost_{max}$ into (14) leads to:

$$\Delta(\sigma_t)+\zeta \cdot cost_{min} \leq \Theta + \zeta \cdot cost_{max} + \sigma_t \cdot \mathbb{E}[\sum_{v \in \mathcal{R}^t} (l_v^{\gamma'^t} - \bar{l})|\sigma_t] \quad (20)$$

Define $\Theta' = \Theta + \zeta \cdot (cost_{max} - cost_{min})$. We have the following:

$$\begin{aligned} \Delta(\sigma_t) &\leq \Theta' + \sigma_t \cdot \mathbb{E}[\sum_{v \in \mathcal{R}^t} (l_v^{\gamma'^t} - \bar{l})|\sigma_t] \\ &\overset{(19)}{\leq} \Theta' - \mathcal{X} \cdot \sigma_t \end{aligned} \quad (21)$$

The upper bound on the average accumulated latency across all the time slots can be obtained:

$$\begin{aligned} \frac{1}{T}\sum_{t=0}^{T-1} \sigma_t &\overset{(21)}{\leq} \frac{\Theta'}{\mathcal{X}} - \frac{\sum_{t=0}^{T-1} \Delta(\sigma_t)}{T\mathcal{X}} \\ &\overset{(11)}{=} \frac{\Theta'}{\mathcal{X}} - \frac{\mathbb{E}[L(\sigma_T) - \mathbb{E}[L(\sigma_0)]}{T\mathcal{X}} \leq \frac{\Theta'}{\mathcal{X}} \end{aligned} \quad (22)$$

Considering the fact that $O(\frac{\Theta'}{\mathcal{X}}) = O(\zeta)$, the averaged accumulate latency across all the time slots of LAO is bounded by $O(\zeta)$. With a lower ζ, the time-averaged accumulate latency is also lower, indicating the acceleration of the stabilization. □

Based on the analysis above, when $\zeta \to \infty$, the solution found by LAO is the optimal one to $\mathcal{P}_1$. However, with a higher value of ζ, the accumulated latency will increase and more time slots are required to stabilize this time-averaged transmission latency (10). The performance analysis presented in this section is also validated and analyzed experimentally in Section 4.

4 Experimental Evaluation

We experimentally evaluated the performance of LAO in different OEDD scenarios by varying different parameters. We conducted all experiments on a Windows-10 machine and a real-world data set.

4.1 Experimental Settings

4.1.1 Benchmark approaches

Three representative approaches are implemented to be compared against LAO:

- *EDD-IP (EI)* [20]: it always provides the optimal solution with the minimum data distribution cost. This approach is an offline approach and is executed in every single time slot for finding a strategy for distributing the corresponding data to the edge cache system always from the cloud.
- *Enhanced-EDD-IP (EEI)*: this approach is the enhanced version of EDD-IP that may source data for destination edge servers from the cloud or from other edge servers in the system. If the long-term transmission latency constraint (4) is fulfilled, EEI aims to minimize the data distribution cost. Otherwise, EEI finds the solution that minimizes the transmission latency to fulfill constraint (4).
- *Latency-Oriented data distribution (LO)*: it always finds the optimal solution with the minimum data distribution latency in each time slot, originated from the optimal solution presented in [34].

4.1.2 Performance metrics

Two metrics are implemented in the experiments to evaluate the performance of LAO:

- Time-averaged transmission latency, calculated according to the left side of (4), the lower the better. We also observe whether the approaches fulfill the long-term transmission latency constraint (4).
- Total transmission cost, calculated with (3) over T, the lower the better.

4.1.3 Data set

All the experiments are conducted on a widely-used real-world dataset[7]. This dataset contains the geographical locations of 125 cellular base stations (edge servers) in Melbourne, Australia. Similar to many studies in the EC environment [4], [20], [35], [36], we measure transmission cost and transmission latency in a generic manner. Specifically, C2E transmission cost is 1 and E2E transmission cost is $ratio$. E2E transmission latency is 1 and C2E transmission latency is 10. In each experiment, the total number of time slots is 200 ($T = 200$). In each experiment, a number of $|V|$ edge servers are randomly selected from the dataset to simulate an edge cache system. The links between edge servers are randomly generated according to $density$ and we ensure that the edge servers in the system constitute a connected edge server network. In each time slot t, data d is randomly decached from each individual edge server according to p_{dec}. The edge servers that still cache d will be the source edge servers in time slot t. In the meantime, a number of the other edge servers randomly request d according to p_{req} and become destination edge servers in time slot t. Then, the four approaches are performed to formulate the corresponding EDD strategies in time slot t.

4.1.4 Experiment parameters

To comprehensively analyze the performance of LAO, we conduct eight sets of experiments, i.e., Set #1 to Set #8, to simulate various real-world edge cache systems and OEDD scenarios.

- Number of edge servers ($|V|$). This parameter decides the size of the edge cache system.
- Edge density ($density$). This parameter represents the density of graph $G(V, E)$ that represents the edge server network in the system. It is calculated by $density = |E|/|V|$.
- Ratio of E2E transmission cost over C2E transmission cost ($ratio$). This parameter simulates different pricing models from edge infrastructure providers for data transmissions.
- Time-averaged maximum transmission latency ($\bar{l}$). Used in Eq. (9), this parameter is the app vendor's

7. https://github.com/swinedge/eua-dataset

TABLE 2: Parameter Settings

	$\|V\|$	$density$	$ratio$	$\bar{l}$	p_{dec}	p_{req}	ζ
Set #1	40	1.0	0.5	5.0	0.6	0.10	1
Set #2	20,30,40,50,60	1.0	0.5	5.0	0.6	0.10	1
Set #3	40	1.0,1.2,1.4,1.6,1.8,2.0	0.5	5.0	0.6	0.10	1
Set #4	40	1.0	0.1,0.3,0.5,0.7,0.9	5.0	0.6	0.10	1
Set #5	40	1.0	0.5	3.0,3.5,4.0,4.5,5.0,5.5,6.0	0.6	0.10	1
Set #6	40	1.0	0.5	5.0	0.8,0.7,0.6,0.5,0.4	0.10	1
Set #7	40	1.0	0.5	5.0	0.6	0.05,0.10,0.15,0.20,0.25	1
Set #8	40	1.0	0.5	5.0	0.6	0.10	100,1,0.01

expected time-averaged maximum transmission latency. A low $\bar{l}$ indicates app vendor's high priority for pursuing low data retrieval latency.

- Data decache possibility (p_{dec}). This parameter determines the possibility for an edge server to remove d from its cache. It simulates the scarcity of app vendor's cache spaces in the system and impacts the number of source edge servers in each time slot.
- Data request possibility (p_{req}). This parameter controls the possibility that an edge server becomes a destination edge server in each time slot. A high p_{req} represents high popularity of d.
- Trade-off parameter (ζ). As discussed in Section 3.2, this parameter is used in (15) to represent the urgency to stabilize the time-averaged transmission latency when (10) was violated.

In the experiments, we vary the values of the seven parameters below to observe their impacts on LAO different OEDD scenarios. Except in Set #1, we vary the value of one setting parameter while fixing the other six parameters in each set of the experiments, as summarized in Table 2. Once a setting parameter is changed, we repeat the experiments for 30 times and obtain the averaged results.

4.2 Experimental Evaluation

4.2.1 Performance Comparison

The results of Set #1 is shown in Fig. 2. Fig. 2(a) depicts that LAO quickly stabilizes the transmission latency and keeps it below $\bar{l}$, fulfilling the long-term transmission latency constraint (4). We can also see that LAO does this at the lowest of the total transmission cost among all the four approaches. LO and EEI can also fulfil (4) over time. LO incurs the lowest time-average transmission latency, followed by LAO with an average performance gap of 19.08%. This is because LO tries to minimize low transmission latency despite the high transmission cost incurred as shown in Fig. 2(b).

Fig. 2(b) demonstrates that LAO significantly outperforms EEI, EI and LO in minimizing transmission cost, by an average of 43.72%, 38.75% and 66.96% over T, respectively, across 200 time slots. The overall transmission cost incurred by EEI is slightly higher than that incurred by EI. This is because EEI tries to fulfill the long-term transmission latency constraint (4) at the price of higher transmission costs. LAO's superior performance demonstrated in Fig. 2 validates the importance of tackling the EDD problem in an online manner.

The results of the time-averaged latency achieved by all the four approaches in Sets #2 - #7 are similar to Fig. 2(a). Thus, we summarize the average results in Table 3. It shows that LAO is second only to LO in all the experiments. Next, we focus on the total transmission costs incurred by the approaches in Sections 4.2.2 - 4.2.7.

4.2.2 Impact of number of edge servers ($|V|$)

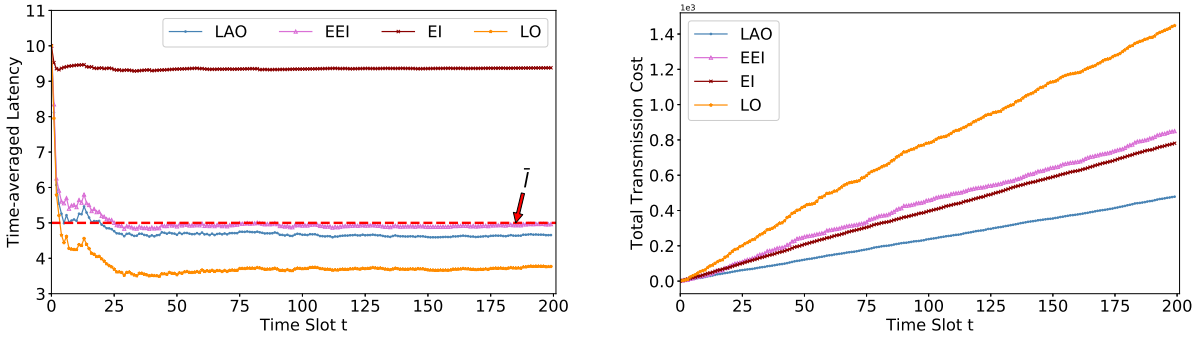
Fig. 3 compares the transmission costs incurred by the four approaches when the size of the edge cache system $|V|$ varies from 20 to 60. It is clear that LAO incurs the lowest total transmission costs over time. On average, the advantages of LAO are 37.69% over EEI, 40.00% over EI and 63.11% over LO. When $|V|$ increases from 20 to 60, more edge servers will request data d on average in each time slot and it takes more time to transmit d to all the destination edge servers in general. LAO can source d for destination edge servers closet to destination edge servers and thus does not incur much more transmission costs when $|V|$ increases. Specifically, when $|V|$ increases from 20 to 60 by 300%, LAO's total transmission cost increases from 307.22 to 719.17 by only 134.09%, much lower than EEI's 179.63% increase from 445.83 to 1,246.66, EI's 210.55% from 437.22 to 1,357.78 and LO's 213.55% from 637.78 to 2,001.67.

4.2.3 Impact of edge density ($density$)

Fig. 4 presents the impacts of $density$ on the total transmission costs. Overall, LAO achieves the lowest total transmission cost. On average, the advantages of LAO are 33.77% over EEI, 36.38% over EI and 58.65% over LO. When $density$ increases from 1.0 to 2.0, the total transmission costs achieved by all the four approaches decrease, by 38.68% from 530.00 to 325.00 for LAO, by 40.83% from 811.20 to 480.00 for EEI, by 20.69% from 762.80 to 605.00 for EI and by 59.42% from 1429.40 to 580. The reason is that, with an increase in $density$, each edge server is connected to more other edge servers in the system on average. This shortens the average distance between edge servers over the edge server network, and data can be sourced to destination edge servers with lower transmission costs.

TABLE 3: Time-averaged Latency (Sets #1 - #7)

	Set #1	Set #2	Set #3	Set #4	Set #5	Set #6	Set #7
LAO	4.65	4.66	4.11	4.62	4.28	4.82	4.63
EEI	4.96	4.83	4.32	4.63	4.53	4.93	4.63
EI	9.38	9.22	9.04	9.31	9.36	9.25	9.24
LO	3.77	3.93	3.46	3.87	3.57	4.16	3.67



(a) Latency over T

(b) Cost over T

Fig. 2: Set #1

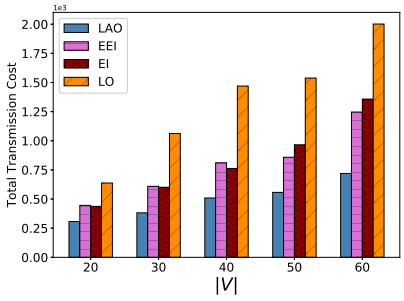


Fig. 3: Cost vs. $|V|$ (Set #2)

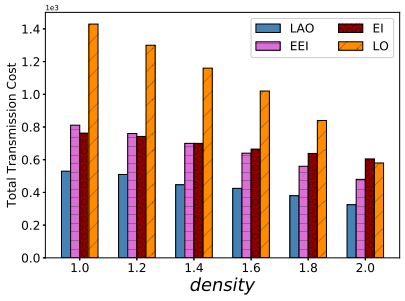


Fig. 4: Cost vs. $density$ (Set #3)

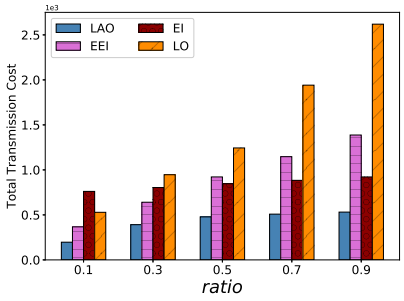


Fig. 5: Cost vs. $ratio$ (Set #4)

4.2.4 *Impact of ratio of E2E transmission cost over C2E transmission cost* $(ratio)$

Fig. 5 compares the total transmission costs when $ratio$ increases from 0.1 to 0.9 in Set #3. Overall, LAO outperforms EEI, EI and LO with large margins, i.e., 54.85% against EEI, 50.62% against EI and 71.07% against LO. The increase in $ratio$ reduces the cost of E2E transmissions over C2E transmissions and consequently increases the total transmission costs incurred all the four approaches. The cost incurred by EI increases only by 21.12% from 761.20 to 922.00 because EI always sources data from the remote cloud. The cost incurred by LO increases the most significantly - from 528.80 to 2619.20 by 395.31%. LO focuses on minimizing transmission latency without considering transmission cost or $ratio$. As a result, more expensive E2E transmissions incurred by a higher $ratio$ immediately increase its transmission cost. EEI's transmission cost also increases with $ratio$, but less significantly than LO, only by 277.89% from 367.20 to 1387.60. As $ratio$ increases, EEI needs more C2E transmissions on average and incurs higher transmission costs. For the same reason, LAO's transmission cost also increases with $ratio$, but is much slower compared with EEI and LO, only by 169.92% from 196.80 to 531.20. This shows that given different $ratio$ values, LAO can always formulate the most cost-effective solutions by finding the best combinations of C2E and E2E transmissions.

4.2.5 *Impact of time-averaged maximum transmission latency* $\bar{l}$

Fig. 6 illustrates the impact of $\bar{l}$ on the total transmission costs in Set #4. Again LAO is the clear winner, with its transmission cost slightly decreases 575.00 to 455.00 by 20.87% when $\bar{l}$ increases from 3.0 to 6.0. EEI's cost decrease

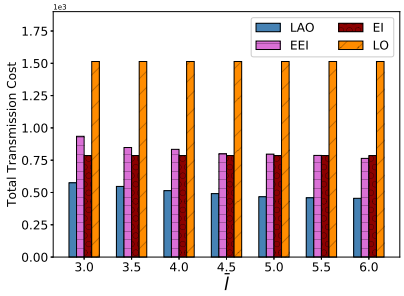


Fig. 6: Cost vs. $\bar{l}$ (Set #5)

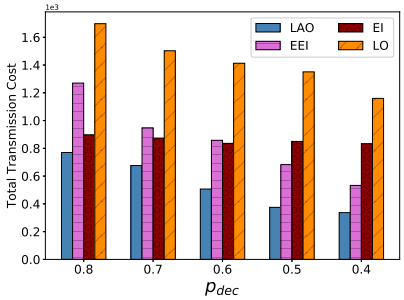


Fig. 7: Cost vs. p_{dec} (Set #6)

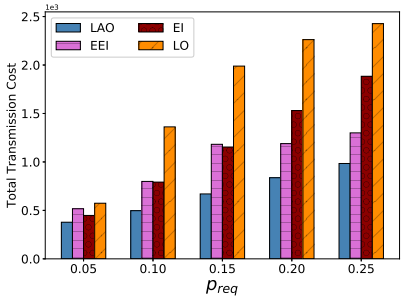
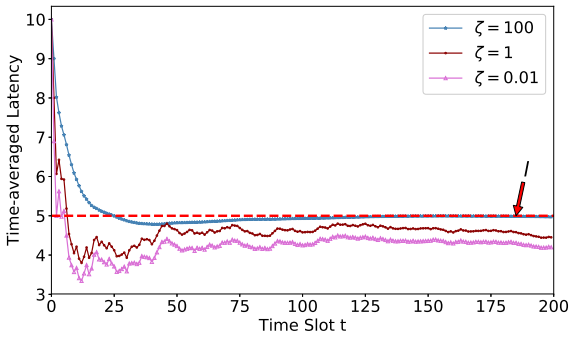
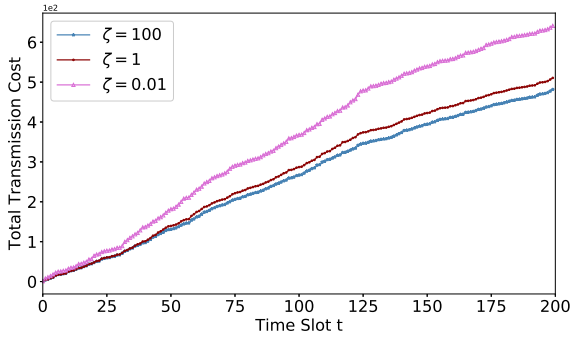


Fig. 8: Cost vs. p_{req} (Set #7)



(a) Time-average Transmission Latency vs. ζ over T



(b) Total Transmission Cost vs. ζ over T

Fig. 9: Latency and Cost achieved by LAO vs. ζ (Set #8)

is also slight, from 935.00 to 764.00 by 18.29%. The increase in $\bar{l}$ relaxes the constraint for low data retrieval latency. LAO and EEI can source data for destination edge servers from edge servers further away to lower their total transmission costs. EI and LO do not consider $\bar{l}$ and thus are not impacted by the value of $\bar{l}$.

4.2.6 *Impact of data decache possibility (p_{dec})*

Fig. 7 shows the impact of p_{dec} on the four approaches in Set #5. Overall, LAO outperforms EEI, EI and LO, by an average of 37.91%, 37.90% and 62.60% respectively. As p_{dec} increases, it is more likely for LAO, EEI and LO to source data from within the system rather than the cloud. Using this advantage, their costs decrease with the increase in p_{dec} significantly, from 770.00 to 337.00 by 56.23% for LAO, from 1270.00 to 533.00 by 58.03% for EEI and from 1698.00 to 1060.00 by 37.57% for LO. The cost incurred by EI decreases only by 7.13% from 898.00 to 834.00 because EI only sources data from the cloud.

4.2.7 *Impact of data request possibility (p_{req})*

Fig. 8 demonstrates the impact of p_{req} on the total transmission costs. Overall, the advantages of LAO are again significant, in terms of the total transmission cost, i.e. 32.51% over EEI, 42.05% over EI and 60/92% over LO. With a higher $request_q$, there are more destination edge servers in each time slot. Thus, total transmission costs achieved by all approaches increase when $request_q$ increases from 0.05 to 0.25.

4.2.8 *Impact of trade-off parameter (ζ)*

Fig. 9 shows the impact of ζ (introduced by (12) into $\mathcal{P}_2$) on LAO. As discussed in Section 3.1, a lower ζ will accelerate the stabilization of the time-averaged transmission latency at a potentially higher transmission cost. It is validated experimentally by the results presented in Fig. 9(b) and Fig. 9(a). As shown in Fig. 9(a), at the very beginning of the experiments, the average latency is high because data d has to be sourced from the remote cloud. When $\zeta = 100$, LAO takes 26 time slots to decrease the average latency to below $\bar{l}$ before it is stabilized. When $\zeta = 1$ and 0.01, LAO takes only 8 and 5 time slots, respectively, to do the same. The fast stabilization comes the price of high cost, as demonstrated in Fig. 9(b). When $\zeta = 0.01$, LAO incurs the highest cost of 641.00 on average, 26.15% and 33.13% higher than when $\zeta = 1$ an $\zeta = 100$, respectively.

4.3 Threats to Validity

4.3.1 *Construct Validity*

The main threats to construct validity of the evaluation are the randomly generated graphs and and the three approaches used for comparison in the experiments. Randomly generated graphs may not always represent real-world scenarios precisely. To minimize this threat, the

graphs are randomly generated in each execution - a total of 100 graphs are generated every time the value of a parameter varies. The three comparison approaches, i.e., EI, EEI and LO, may not suffice to evaluate LAO comprehensively. To minimize this threat, we conducted comprehensive experiments by varying seven experimental parameters in Table 2 to evaluate the performance of LAO in various OEDD scenarios. In this way, we could evaluate LAO by comparison against EI, EEI and LO, and also by the impacts of those seven experimental parameters.

4.3.2 Threats to Internal Validity

Whether the experiment setting favors LAO over other approaches is the main threat to the internal validity. To tackle this threat, we simulated various OEDD scenarios by changing seven parameters. This way, we could comprehensively and fairly compare the performance of different approaches. In addition, we repeated the experiments for 30 times to obtain the averaged results when a parameter was changed. Then biased results obtained in experiments are neutralized.

4.3.3 Threats to External Validity

LAO's generalization and application in different OEDD scenarios are the main threat to the external validity of our evaluation. To reduce this threat, we modeled the OEDD problem and evaluated all approaches in a generic manner. For example, we used the number of hops to measure the latency and the unit transmission cost to measure the C2E transmission cost and E2E transmission cost, similar to [20]. Therefore, we can easily interpret the evaluation results with specific latency and cost models, e.g. Amazon or Google's price model. Furthermore, the real-world EUA data set was adopted to execute the experiments. We also varies the size and the complexity of the OEDD problem by changing the parameters in experimental settings. In this way, the representativeness and comprehensiveness of the experimental evaluation can be improved, and the threat to external validity is mitigated.

5 Related Work

The evolution of broad and mobile communications in the past two decades has continuously lowered the expenses of bandwidth and latency has become the main impediment to improving user-perceived performance [11]. Amazon estimates that every 100ms increase in latency cuts its sales and profits by 1% [37].

The key to low data retrieval latency is to cache the right data within end-users' close reach, especially for edge cache systems that must accommodate end-users' dynamic demands [38]. There is a large body of work on how to determine the right data to cache dynamically through user demand prediction [39]. The key performance metric is the cache hit ratio. It measures the percentage of data requests that can be served by the cache system. The performance improvement of a cache system relies on the increase in the accuracy of user demand prediction. The other main method for reducing data retrieval latency is to shorten the distance between data cache and end-users. To achieve this goal, large-scale content delivery infrastructures like content delivery networks (CDNs) have been built in the cloud. Data replicas can be cached on up to thousands of distributed CDN nodes (servers) around the globe [40] to serve end-users in the corresponding regions [23]. CDNs have facilitated further advances in the application of user demand prediction techniques to cache systems. The key idea is to reduce or minimize the cache hit distance that measures the distance between end-users and the CDN nodes serving them [23]. However, there is a fundamental limit on the caching performance improvements that can be achieved through the re-engineering and expansion of CDNs [41]. End-users still have to retrieve data from the public internet and it is difficult to predict, control or reduce their data retrieval latency [11]. It is the main barrier to further performance improvement of data caching for latency-sensitive applications like gaming and VR.

Edge computing (EC) enables 5G to break that barrier by allowing data to be cached on edge servers deployed at base stations [42]. The caching system facilitated by edge servers reduces the physical distance between end-users and data to hundreds of meters [9]. Edge data caching has attracted a great deal of attention in very recent years [43]. Similar to studies of conventional cloud cache systems like CDN, existing studies of edge cache systems try to achieve various optimization goals from the edge infrastructure provider's perspective, e.g., maximizing caching benefits [4], [16] and minimizing data retrieve latency [15], [17], [44]. The approaches proposed in these studies fully leverage the short physical distance between data and end-users in edge cache systems. However, the impacts of the long distance between the remote cloud and edge servers on edge cache systems have been ignored completely, including the potentially high costs and latency incurred by the data transmission from the remote cloud server. Xia et al. studied these impacts and tried to solve the edge data distribution problem in an offline manner [20]. However, their approach always have to fetch data from the remote cloud server onto edge servers and incurs impractically high costs and latency in the long term, especially for data whose popularity often vary at different locations over time [21], as demonstrated in Section 4. To help app vendors utilize edge cache systems in the long term, this paper tackles the online edge data distribution problem as the first attempt. This allows our approach to source data from within the edge cache system. The experimental results presented and discussed in Section 4 demonstrated its superior performance in minimizing data transmission costs and stabilizing data transmission latency over time.

6 Conclusion

In this paper, we studied the online edge data distribution (OEDD) problem in the Edge Computing (EC) environment from the app vendor's perspective. We first formulated the OEDD problem and proved its $\mathcal{NP}$-hardness. Then, we proposed a new Latency-Aware OEDD algorithm, named LAO, that formulates cost-effective EDD strategies online over time. We evaluated LAO's performance both theoretically and experimentally. This research builds a solid foundation for enabling edge cache systems in the real world and opens up a number of new research directions along the line.

In our future work, we will investigate the distribution of streaming data and try to fault-tolerate the data distribution process.

ACKNOWLEDGEMENT

This research is partially funded by Australian Research Council Discovery Projects No. DP180100212, DP200102491 and Laureate Fellowship FL190100035.

REFERENCES

[1] CISCO, *Cisco Edge-to-Enterprise IoT Analytics for Electric Utilities Solution Overview*, February 1, 2018. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/big-data/solution-overview-c22-740248.html

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2019.

[4] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2020.

[5] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[6] Y. Ma, X. Liu, S. Zhang, R. Xiang, Y. Liu, and T. Xie, "Measurement and analysis of mobile web cache performance," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 691–701.

[7] K. Ji, G. Quan, and J. Tan, "Asymptotic miss ratio of lru caching with consistent hashing," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2018, pp. 450–458.

[8] R. Halalai, P. Felber, A.-M. Kermarrec, and F. Taïani, "Agar: A caching system for erasure-coded data," in *37th IEEE International Conference on Distributed Computing Systems*, 2017, pp. 23–33.

[9] N. Garg, M. Sellathurai, V. Bhatia, B. Bharath, and T. Ratnarajah, "Online content popularity prediction and learning in wireless edge caching," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1087–1100, 2019.

[10] Y. Wang, B. Veeravalli, and C.-K. Tham, "On data staging algorithms for shared data accesses in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 4, pp. 825–838, 2012.

[11] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: the virtue of gentle aggression," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013, pp. 159–170.

[12] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing cache data integrity in the edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1210–1223, 2021.

[13] Q. Li, Y. Zhang, A. Pandharipande, Y. Xiao, and X. Ge, "Edge caching in wireless infostation networks: Deployment and cache content placement," in *IEEE Conference on Computer Communications Workshops*. IEEE, 2019, pp. 1–6.

[14] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1863–1876, 2016.

[15] T. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2018.

[16] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *38th IEEE International Conference on Distributed Computing Systems*. IEEE, 2018, pp. 388–399.

[17] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *37th IEEE International Conference on Distributed Computing Systems*. IEEE, 2017, pp. 276–286.

[18] Y. Li, H. Ma, L. Wang, S. Mao, and G. Wang, "Optimized content caching and user association for edge computing in densely deployed heterogeneous networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[19] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes, and http chunking in web search," in *Velocity Web Performance and Operations Conference*. oreilly, 2009.

[20] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 31–44, 2021.

[21] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 915–929, 2018.

[22] J. Gao, S. Zhang, L. Zhao, and X. S. Shen, "The design of dynamic probabilistic caching with time-varying content popularity," *IEEE Transactions on Mobile Computing*, 2020.

[23] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1473–1499, 2015.

[24] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[25] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4514–4526, 2018.

[26] Y. Jiang, M. Ma, M. Bennis, F.-C. Zheng, and X. You, "User preference learning-based edge caching for fog radio access network," *IEEE Transactions on Communications*, vol. 67, no. 2, pp. 1268–1283, 2018.

[27] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[28] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "A novel mobile edge network architecture with joint caching-delivering and horizontal cooperation," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 19–31, 2019.

[29] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, "Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[30] Y. Huang, J. Zhang, J. Duan, B. Xiao, F. Ye, and Y. Yang, "Resource allocation and consensus of blockchains in pervasive edge computing environments," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[31] X. Xia, F. Chen, J. Grundy, M. Abdelrazek, H. Jin, and Q. He, "Constrained app data caching over edge server graphs in edge computing environment," *IEEE Transactions on Services Computing*, pp. 1–1, 2021.

[32] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE Conference on Computer Communications*, 2018, pp. 207–215.

[33] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[34] F. Tang, H. Zhang, and L. T. Yang, "Multipath cooperative routing with efficient acknowledgement for leo satellite networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 179–192, 2018.

[35] G. Cui, Q. He, F. Chen, Y. Zhang, H. Jin, and Y. Yang, "Interference-aware game-theoretic device allocation for mobile edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[36] B. Li, Q. He, G. Cui, X. Xia, F. Chen, H. Jin, and Y. Yang, "Read: Robustness-oriented edge application deployment in edge computing environment," *IEEE Transactions on Services Computing*, 2020.

[37] S. Puzhavakath Narayanan, Y. S. Nam, A. Sivakumar, B. Chandrasekaran, B. Maggs, and S. Rao, "Reducing latency through page-aware management of web objects by content delivery networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1, pp. 89–100, 2016.

[38] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374–398, 2003.

[39] W. Ali, S. M. Shamsuddin, A. S. Ismail *et al.*, "A survey of web caching and prefetching," *Int. J. Advance. Soft Comput. Appl*, vol. 3, no. 1, pp. 18–44, 2011.

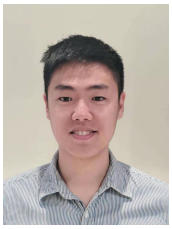
[40] Microsoft. (2020) Content delivery networks (cdns). [Online]. Available: https://docs.microsoft.com/en-us/microsoft-365/enterprise/content-delivery-networks

[41] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, "Moving beyond end-to-end path information to optimize cdn performance," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, 2009, pp. 190–201.

[42] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[43] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2525–2553, 2019.

[44] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2017.



Xiaoyu Xia received his Master degree from The University of Melbourne, Australia in 2015. He is a PhD candidate at Deakin University. His research interests include edge computing, parallel and distributed computing, service computing, software engineering and cloud computing.

Feifei Chen received her PhD degree from Swinburne University of Technology, Australia in 2015. She is a lecturer at Deakin University. Her research interests include software engineering, cloud computing and green computing.

Qiang He received his first PhD degree from Swinburne University of Technology, Australia, in 2009 and his second PhD degree in computer science and engineering from Huazhong University of Science and Technology, China, in 2010. He is an Associate Professor at Swinburne. His research interests include service computing, software engineering, cloud computing and edge computing. More details about his research can be found at https://sites.google.com/site/heqiang/.

John C. Grundy received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently Australian Laureate Fellow and a professor of software engineering at Monash University, Melbourne, Australia. He is an associate editor of the IEEE Transactions on Software Engineering, the Automated Software Engineering Journal, and IEEE Software. His current interests include domain-specific v isual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at https://sites.google.com/site/johncgrundy/.

Mohamed Abdelrazek is an Associate Professor of Software Engineering and IoT at Deakin University. Before joining Deakin University in 2015, he worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). Before 2010, he was the head of software development department at Microtech. More details about his research can be found at https://sites.google.com/site/mohamedalmorsy/.

Jun Shen is a Associate Professor at the University of Wollongong in Wollongong, NSW Australia. His expertise is on web services and Semantic Web. He has been an editor, PC chair, guest editor, and a PC member for numerous journals and conferences published by the IEEE, ACM, Elsevier, and Springer. From 2007, he was a chair of Education Chapter of IEEE NSW section. He is a senior member of the IEEE.

Athman Bouguettaya is a Professor in the School of Computer Science at University of Sydney, Australia. He received his PhD in Computer Science from the University of Colorado at Boulder (USA) in 1992. He was previously Science Leader in Service Computing at CSIRO ICT Centre, Canberra. Australia. Before that, he was a tenured faculty member and Program director in the Computer Science department at Virginia Polytechnic Institute and State University (commonly known as Virginia Tech) (USA). He is or has been on the editorial boards of several journals including, the IEEE Transactions on Services Computing, ACM Transactions on Internet Technology, the International Journal on Next Generation Computing and VLDB Journal. He was the recipient of several federally competitive grants in Australia (e.g., ARC) and the US (e.g., NSF, NIH). He is a Fellow of the IEEE and a Distinguished Scientist of the ACM.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. Jin received his PhD in computer engineering from HUST in 1994. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

5.3 Summary

In this chapter, we formulate the edge data distribution problem in the edge computing environment from the app vendor's perspective to minimize the cost incurred by data distribution. We prove that the edge data distribution problem is NP-hard. In **Chapter 5.1**, we study this edge data distribution problem in quasi-static scenarios. To solve this problem, we first propose an optimal solution named EDD-IP, and then provide a tree-based approximation approach. With consideration of the dynamics in the edge computing environment, we further study this data distribution problem in an online manner in **Chapter 5.2**. Without requiring complete information in advance, we design an online approach based on Lyapunov Theory. The results of experiments conducted on a real-world dataset demonstrate that our proposed approach can help app vendors formulate cost-effective edge data distribution strategies in an online manner. This research builds a solid foundation for enabling edge cache systems in the real world and opens up a number of new research directions along the line.

Chapter 6

Conclusions and Future Work

This thesis mainly discusses the data caching problems in edge computing, including edge data placement problem, edge data replacement problem and edge data distribution problem. Those problems are identified and the corresponding solutions are proposed in **Chapters 3 - 5**, respectively. In this chapter, we summarize the contributions within the body of this work. After that, we indicate several open issues for future research.

6.1 Conclusions

The main contributions of this thesis are as follows:

1. We make the first attempt to investigate the edge data placement problems from an app vendor's perspective in the edge computing environment. We first study the individual data placement problem with the aim to minimize the data caching cost while ensuring all users can retrieve data from their nearby edge servers. Considering that an app vendor normally have multiple data to be cached, we further study the multi-data placement problem, aiming to minimize

all users' data retrieval latency with limited cache spaces on edge servers.

2. We tackle the edge data replacement problem for app vendors as the first attempt in the dynamic edge computing environment. To minimize app vendors' system cost while ensuring users' low data retrieval latency in a long term, we propose a Lyapunov-based online approach, named CEDC-O to solve this problem without future information. Considering the cost-effectiveness, we propose an online frame named OL-MEDC, embedded an approximation algorithm to improve in effectiveness and efficiency for solving this edge data replacement problem.

3. As the first attempt, we study the edge data distribution problem from an app vendor's perspective with the aim to minimize the cost incurring the data distribution from the cloud to edge servers. We first propose a two-phase algorithm based on the Rooted Minimum Steiner Tree problem to ensure that this edge data distribution problem can be solved effectively and efficiently in quasi-static scenarios. In the dynamic edge computing scenarios, the edge data distribution problem becomes more complex due to the uncertainty of future information. Thus, we design an online approach to solve this edge data distribution in each time slot without complete information about future dynamics.

6.2 Future Work

Although the proposed methods solved some data caching problems in various edge computing scenarios, there are still some problems that need to be further addressed. This section discusses the open issues as the extensions of the presented work in the

thesis.

Edge data placement. We have discussed the edge data placement problem in **Chapter 3**, and investigated both individual data placement problem and multi-data placement problem from app vendors' perspective. However, there is an implicit assumption that all the cached data are always reliable and trustworthy. With edge servers deployed in a geographically distributed manner, the EC environment is highly dynamic and volatile, subject to various intentional and accidental corruptions as well as many security threats. Due to edge servers' limited resources, they are much less powerful and reliable than cloud servers deployed in mega-size data centers, and thus are more vulnerable to intentional and accidental corruptions. Hence, edge servers must not be assumed reliable or trustworthy all the time. Taking into account the various potential faults, an app vendor must deploy its application instances on the edge servers to collectively deliver a robust service to the users in that area. Therefore, we would like to investigate the robust edge data placement problem in our future work, considering the user coverage, data retrieval latency and robustness.

Edge data replacement. Chapter 4 shows our two proposed approaches to tackle the edge data replacement problem with the aims to minimize the system cost and maximize the system revenue. Similar to the limitations mentioned in edge data distribution, the wireless communication between edge servers and users are not considered in this thesis, which can significantly impact users' quality of service. In this case, the data rates received by users cannot be guaranteed. In addition, the approaches proposed in **Chapter 4** always take the same processes whatever the data request pattern is, e.g. data block, data file or data streaming. In the future work, we would like to investigate this edge data replacement problem with consideration

of wireless communication, and design different approaches for updating data in edge servers' caches with specific data patterns.

Edge data distribution. In **Chapter 5**, we discuss the edge data distribution problems and propose two approaches for app vendors in both quasi-static and dynamic edge computing systems. However, the problems only studied focus on the single data distribution in edge computing systems. Multi-data distributed simultaneously could complicate this edge data distribution significantly. In addition, one limitation is that the transmission latency from edge servers to users is ignored. In the edge computing systems, users, especially mobile users, needs to receive data via wireless communication, in which the interference among users could impact users' data rates severely. Thus, in the future work, we would like to investigate the edge data distribution in a dynamic multi-access edge computing environment for providing cost-effective edge data distribution strategies.

Bibliography

[1] Arteaga, D., Cabrera, J., Xu, J., Sundararaman, S., Zhao, M.: Cloudcache: On-demand flash cache management for cloud computing. In: 14th USENIX Conference on File and Storage Technologies. pp. 355–369 (2016)

[2] Badawy, A.H.A., Yessin, G., Narayana, V., Mayhew, D., El-Ghazawi, T.: Optimizing thin client caches for mobile cloud computing: Design space exploration using genetic algorithms. Concurrency and Computation: Practice and Experience **29**(11), 1–13 (2017)

[3] Banerjee, B., Kulkarni, A., Seetharam, A.: Greedy caching: An optimized content placement strategy for information-centric networks. Computer Networks **140**, 78–91 (2018)

[4] Berger, D.S., Sitaraman, R.K., Harchol-Balter, M.: Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In: 14th USENIX Symposium on Networked Systems Design and Implementation. pp. 483–498 (2017)

[5] Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on

Mobile cloud computing. pp. 13–16 (2012)

[6] Breitbach, M., Schäfer, D., Edinger, J., Becker, C.: Context-aware data and task placement in edge computing environments. In: 2019 IEEE International Conference on Pervasive Computing and Communications. pp. 1–10. IEEE (2019)

[7] Cai, H., Xu, B., Jiang, L., Vasilakos, A.V.: Iot-based big data storage systems in cloud computing: perspectives and challenges. IEEE Internet of Things Journal **4**(1), 75–87 (2016)

[8] Cao, X., Zhang, J., Poor, H.V.: An optimal auction mechanism for mobile edge caching. In: 38th IEEE International Conference on Distributed Computing Systems. pp. 388–399 (2018)

[9] Chen, L., Zhou, S., Xu, J.: Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. IEEE/ACM Transactions on Networking **26**(4), 1619–1632 (2018)

[10] Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Transactions on Networking **24**(5), 2795–2808 (2016)

[11] Chen, Y., Zhang, N., Zhang, Y., Chen, X., Wu, W., Shen, X.S.: Energy efficient dynamic offloading in mobile edge computing for internet of things. IEEE Transactions on Cloud Computing (2019). https://doi.org/10.1109/TCC.2019.2898657

[12] Cho, K., Lee, M., Park, K., Kwon, T.T., Choi, Y., Pack, S.: Wave: Popularity-based and collaborative in-network caching for content-oriented networks. In: IEEE Conference on Computer Communications Workshops. pp. 316–321 (2012)

[13] Dehghan, M., Jiang, B., Seetharam, A., He, T., Salonidis, T., Kurose, J., Towsley, D., Sitaraman, R.: On the complexity of optimal request routing and content caching in heterogeneous cache networks. IEEE/ACM Transactions on Networking **25**(3), 1635–1648 (2017)

[14] Deng, S., Xiang, Z., Taheri, J., Mohammad, K.A., Yin, J., Zomaya, A., Dustdar, S.: Optimal application deployment in resource constrained distributed edges. IEEE Transactions on Mobile Computing (2020). https://doi.org/10.1109/TMC.2020.2970698

[15] Deng, S., Xiang, Z., Zhao, P., Taheri, J., Gao, H., Yin, J., Zomaya, A.Y.: Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method. IEEE Transactions on Industrial Informatics **16**(9), 6103–6113 (2020)

[16] Drolia, U., Guo, K., Tan, J., Gandhi, R., Narasimhan, P.: Cachier: Edge-caching for recognition applications. In: 37th IEEE International Conference onDistributed Computing Systems. pp. 276–286 (2017)

[17] Elhardt, K., Bayer, R.: A database cache for high performance and fast restart in database systems. ACM Transactions on Database Systems **9**(4), 503–525 (1984)

[18] ETSI, M.: Mobile edge computing - introductory technical white paper (2014)

[19] Forecast, C.V.: Cisco visual networking index: Global mobile data traffic forecast update. 2017–2022 White Paper (Feb,2019)

[20] Gai, K., Qiu, L., Chen, M., Zhao, H., Qiu, M.: Sa-east: security-aware efficient data transmission for its in mobile heterogeneous cloud computing. ACM Transactions on Embedded Computing Systems **16**(2), 1–22 (2017)

[21] Gharaibeh, A., Khreishah, A., Ji, B., Ayyash, M.: A provably efficient online collaborative caching algorithm for multicell-coordinated systems. IEEE Transactions on Mobile Computing **15**(8), 1863–1876 (2016)

[22] Guo, H., Liu, J.: Collaborative computation offloading for multi-access edge computing over fiber-wireless networks. IEEE Transactions on Vehicular Technology **67**(5), 4514–4526 (2018)

[23] Halalai, R., Felber, P., Kermarrec, A.M., Taïani, F.: Agar: A caching system for erasure-coded data. In: 37th IEEE International Conference onDistributed Computing Systems. pp. 23–33 (2017)

[24] Tamoor-ul Hassan, S., Samarakoon, S., Bennis, M., Latva-Aho, M., Hong, C.S.: Learning-based caching in cloud-aided wireless networks. IEEE Communications Letters **22**(1), 137–140 (2018)

[25] He, Q., Cui, G., Zhang, X., Chen, F., Deng, S., Jin, H., Li, Y., Yang, Y.: A game-theoretical approach for user allocation in edge computing environment. IEEE Transactions on Parallel and Distributed Systems **31**(3), 515–529 (2019)

[26] Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing—a key technology towards 5g. ETSI white paper **11**(11), 1–16 (2015)

[27] Karamchandani, N., Niesen, U., Maddah-Ali, M.A., Diggavi, S.N.: Hierarchical coded caching. IEEE Transactions on Information Theory **62**(6), 3212–3229 (2016)

[28] Lai, P., He, Q., Abdelrazek, M., Chen, F., Hosking, J., Grundy, J., Yang, Y.: Optimal edge user allocation in edge computing with variable sized vector bin packing. In: International Conference on Service-Oriented Computing. pp. 230–245 (2018)

[29] Li, C., Bai, J., Tang, J.: Joint optimization of data placement and scheduling for improving user experience in edge computing. Journal of Parallel and Distributed Computing **125**, 93–105 (2019)

[30] Li, Y., Xie, H., Wen, Y., Zhang, Z.L.: Coordinating in-network caching in content-centric networks: Model and analysis. In: 33rd IEEE International Conference on Distributed Computing Systems. pp. 62–72 (2013)

[31] Liu, Y., Dong, M., Ota, K., Liu, A.: Activetrust: Secure and trustable routing in wireless sensor networks. IEEE Transactions on Information Forensics and Security **11**(9), 2013–2027 (2016)

[32] Luo, G., Zhou, H., Cheng, N., Yuan, Q., Li, J., Yang, F., Shen, X.: Software-defined cooperative data sharing in edge computing assisted 5g-vanet. IEEE Transactions on Mobile Computing **20**(3), 1212–1229 (2021)

[33] Ma, K., Yang, B., Yang, Z., Yu, Z.: Segment access-aware dynamic semantic cache in cloud computing environment. Journal of Parallel and Distributed Computing **110**, 42–51 (2017)

[34] Ma, X., Zhou, A., Zhang, S., Wang, S.: Cooperative service caching and workload scheduling in mobile edge computing. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications. pp. 2076–2085. IEEE (2020)

[35] Maddah-Ali, M.A., Niesen, U.: Fundamental limits of caching. IEEE Transactions on Information Theory **60**(5), 2856–2867 (2014)

[36] Maddah-Ali, M.A., Niesen, U.: Decentralized coded caching attains order-optimal memory-rate tradeoff. IEEE/ACM Transactions on Networking **23**(4), 1029–1040 (2015)

[37] Ouyang, T., Li, R., Chen, X., Zhou, Z., Tang, X.: Adaptive user-managed service placement for mobile edge computing: An online learning approach. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. pp. 1468–1476. IEEE (2019)

[38] Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. In: Computer graphics forum. vol. 26, pp. 80–113. Wiley Online Library (2007)

[39] Podlipnig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. ACM Computing Surveys **35**(4), 374–398 (2003)

[40] Poularakis, K., Llorca, J., Tulino, A.M., Taylor, I., Tassiulas, L.: Joint service placement and request routing in multi-cell mobile edge computing networks. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. pp. 10–18 (2019)

[41] Rodrigues, T.G., Suto, K., Nishiyama, H., Kato, N.: Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control. IEEE Transactions on Computers **66**(5), 810–819 (2016)

[42] Schadt, E.E., Linderman, M.D., Sorenson, J., Lee, L., Nolan, G.P.: Computational solutions to large-scale data management and analysis. Nature reviews genetics **11**(9), 647–657 (2010)

[43] Shafiq, M.Z., Khakpour, A.R., Liu, A.X.: Characterizing caching workload of a large commercial content delivery network. In: 35th Annual IEEE International Conference on Computer Communications. pp. 1–9 (2016)

[44] Smith, A.J.: Disk cache—miss ratio analysis and design considerations. ACM Transactions on Computer Systems **3**(3), 161–203 (1985)

[45] Sourlas, V., Paschos, G.S., Flegkas, P., Tassiulas, L.: Mobility support through caching in content-based publish/subscribe networks. In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. pp. 715–720 (2010)

[46] Stenstrom, P.: A survey of cache coherence schemes for multiprocessors. Computer **23**(6), 12–24 (1990)

[47] Tran, T., Pompili, D.: Adaptive bitrate video caching and processing in mobile-edge computing networks. IEEE Transactions on Mobile Computing pp. 1–15 (2018)

[48] Tran, T.X., Hosseini, M.P., Pompili, D.: Mobile edge computing: Recent efforts and five key research directions. IEEE COMSOC MMTC Commun.-Frontiers **12**(4), 29–33 (2017)

[49] Uddin, M.Y.S., Venkatasubramanian, N.: Edge caching for enriched notifications delivery in big active data. In: 38th IEEE International Conference on Distributed Computing Systems. pp. 696–705 (2018)

[50] Wang, C., Liang, C., Yu, F.R., Chen, Q., Tang, L.: Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. IEEE Transactions on Wireless Communications **16**(8), 4924–4938 (2017)

[51] Wang, C.Y., Lim, S.H., Gastpar, M.: Information-theoretic caching: Sequential coding for computing. IEEE Transactions on Information Theory **62**(11), 6393–6406 (2016)

[52] Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J., Wang, W.: A survey on mobile edge networks: Convergence of computing, caching and communications. IEEE Access **5**, 6757–6779 (2017)

[53] Wang, X., Wang, K., Wu, S., Di, S., Jin, H., Yang, K., Ou, S.: Dynamic resource scheduling in mobile edge cloud with cloud radio access network. IEEE Transactions on Parallel and Distributed Systems **29**(11), 2429–2445 (2018)

[54] Wang, Y., Veeravalli, B., Tham, C.K.: On data staging algorithms for shared data accesses in clouds. IEEE Transactions on Parallel and Distributed Systems **24**(4), 825–838 (2012)

[55] Wang, Y., Li, Z., Tyson, G., Uhlig, S., Xie, G.: Optimal cache allocation for content-centric networking. In: 21st IEEE International Conference on Network Protocols. pp. 1–10 (2013)

[56] Xie, J., Guo, D., Shi, X., Cai, H., Qian, C., Chen, H.: A fast hybrid data sharing framework for hierarchical mobile edge computing. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications. pp. 2609–2618. IEEE (2020)

[57] Xie, J., Qian, C., Guo, D., Li, X., Shi, S., Chen, H.: Efficient data placement and retrieval services in edge computing. In: 2019 IEEE 39th International Conference on Distributed Computing Systems. pp. 1029–1039. IEEE (2019)

[58] Xu, J., Chen, L., Zhou, P.: Joint service caching and task offloading for mobile edge computing in dense networks. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications. pp. 207–215. IEEE (2018)

[59] Yang, L., Zhang, H., Li, M., Guo, J., Ji, H.: Mobile edge computing empowered energy efficient task offloading in 5g. IEEE Transactions on Vehicular Technology **67**(7), 6398–6409 (2018)

[60] Yannuzzi, M., van Lingen, F., Jain, A., Parellada, O.L., Flores, M.M., Carrera, D., Pérez, J.L., Montero, D., Chacin, P., Corsaro, A., et al.: A new era for cities with fog computing. IEEE Internet Computing **21**(2), 54–67 (2017)

[61] You, C., Huang, K., Chae, H., Kim, B.H.: Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Transactions on Wireless Communications **16**(3), 1397–1411 (2017)

[62] Zhang, K., Leng, S., He, Y., Maharjan, S., Zhang, Y.: Cooperative content caching in 5g networks with mobile edge computing. IEEE Wireless Communications **25**(3), 80–87 (2018)

[63] Zhang, Q., Zhang, Q., Shi, W., Zhong, H.: Firework: Data processing and sharing for hybrid cloud-edge analytics. IEEE Transactions on Parallel and Distributed Systems **29**(9), 2004–2017 (2018)

[64] Zhang, X., Zhu, Q.: Collaborative hierarchical caching over 5g edge computing mobile wireless networks. In: 2018 IEEE International Conference on Communications. pp. 1–6. IEEE (2018)

[65] Zhang, X., Zhu, Q.: Hierarchical caching for statistical qos guaranteed multimedia transmissions over 5g edge computing mobile wireless networks. IEEE Wireless Communications **25**(3), 12–20 (2018)

[66] Zhao, H., Deng, S., Liu, Z., Yin, J., Dustdar, S.: Distributed redundancy scheduling for microservice-based applications at the edge. IEEE Transactions on Services Computing (2020)

[67] Zhou, J., Fan, J., Wang, J., Jia, J.: Dynamic service deployment for budget-constrained mobile edge computing. Concurrency and Computation: Practice and Experience **31**(24), 1–16 (2019)