When Edge Caching Meets a Budget: Near Optimal Service Delivery in Multi-Tiered Edge Clouds

Qiufen Xia, *Member, IEEE*, Wenhao Ren, Zichuan Xu, *Member, IEEE*, Xin Wang, *Senior Member, IEEE*, and Weifa Liang, *Senior Member, IEEE*

Abstract—More and more artificial intelligence (AI) applications, such as virtual reality (VR) and video analytics, are rapidly progressing towards enterprise and end-users with the promise of bringing immersive experience. Driven by the desire to improve users' experience and promote business scenarios, such AI applications have unprecedented requirements for ultra-low latency as well as abundant computing resource in networks. Data centers in the core network can meet these demands by deploying various AI services and providing abundant resources. However, data transmission delay from data centers to end-users is too time-consuming because of traffic congestion in the core network, which compromises the performance of the AI applications. 5G and edge computing are emerging technologies to guarantee the timeliness for the delay-sensitive applications. The delay experienced by AI users can be significantly reduced, by 'caching' various services that are initially deployed at data centers to cloudlets in edge networks. Although ubiquitous edge service caching is always preferable for improving user experiences, it is impractical to cache all services from data centers to edge cloudlets, due to often limited caching budget of service providers and resource capacity constraints of cloudlets. Therefore, a service provider has to cautiously decide how many instances of a service can be cached, and where to cache the service instances. In this article, we investigate a fundamental problem of *service caching* from remote data centers to edge cloudlets in a multi-tiered edge cloud network. We first develop two approximation algorithms with approximation ratios to solve the problem for users demanding a single type of service. We then devise an efficient heuristic to solve the problem that users require different types of services. We finally conduct extensive experiments on a real test-bed to evaluate the performance of the proposed algorithms, and experimental results demonstrate that our algorithms can outperform some existing algorithms significantly.

Index Terms—Mobile edge clouds, service caching, capacitated k-median, approximation algorithms

1 Introduction

With the progress of AI techniques, we are seeing increasing delay-sensitive (e.g., virtual reality (VR)) services and applications in practical use, ranging from entertainment, gaming, health-care simulations, to profiling future buildings. For example, Disney's latest live-action remake of film *The Lion King* has been a hot topic, because it7 beats past industry expectations by leveraging the VR technology.[1] To offer better immersive experience, the VR services usually generate considerable amount of data, including high resolution videos, render data, human-computer interactions, and operations. As the VR devices typically have limited storage and computing capabilities, many VR service providers leverage remote data centers with abundant resources to process such large volume of data. However, a serious problem encountered by current VR applications is that the conversion from design data generated by VR services in data centers (such as building information modeling service) to VR displays is too time-consuming, especially when the core network is becoming increasingly congested. With the emergence of 5G technology, increasing demands of VR applications for ultra-low delay interactions are driving the moving of services and resources from data centers to the network edge within the proximity of users. Specifically, by caching VR services into edge cloudlets, the delay suffered by users that demand VR services can be significantly reduced [24], thereby improving user experience. Here, a cloudlet is a trusted, resource-rich computer or cluster of computers that's well-connected to the Internet and available for use by nearby mobile devices [22]. Since most cloudlets have limited computing resources, they often work together with data centers to accomplish the VR tasks, while not exceeding constraints of their computing resource capacity and users' waiting delay requirements.

In this paper, we investigate the service caching problem for delay-sensitive applications and requests in a multi-tiered edge cloud network consisting of both remote data centers

1. https://www.videomaker.com/news/the-lion-king-is-revolutionizing-the-future-of-cinema-with-vr/

- *Qiufen Xia and Wenhao Ren are with the International School of Information Science and Engineering, Dalian University of Technology, Dalian 116024, China. E-mail: qiufenxia@dlut.edu.cn, hangvane@mail.dlut.edu.cn.*
- *Zichuan Xu is with the School of Software, and the Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian University of Technology, Dalian 116024, China. E-mail: z.xu@dlut.edu.cn.*
- *Xin Wang is with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794 USA. E-mail: xwang@ece.sunysb.edu.*
- *Weifa Liang is with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong. E-mail: weifa.liang@cityu.edu.hk.*

Manuscript received 15 July 2020; revised 10 June 2021; accepted 14 June 2021. Date of publication 22 June 2021; date of current version 9 Dec. 2022.
(Corresponding author: Zichuan Xu.)
Digital Object Identifier no. 10.1109/TSC.2021.3091462

and cloudlets, where each type of service is cached into cloudlets to reduce users' waiting delay and service providers have a budget for caching these service. Solving the problem faces the following three challenges. The fundamental challenge is how many service instances (replicas) should be cached in cloudlets for each type of services, considering the limited resource capacities of edge cloudlets and the caching budget for services. Another challenge is where to cache the instances for each type of service and dispatch users' requests, such that the average delay experienced by users can be minimized. The third challenge is how to implement requests with various packet rates and different demanded service types. In particular, different types of services require various resource demands, and have different processing costs and delays. Finding a performance-guaranteed solution to the service caching problem requires non-trivial trade-offs among the costs and delays.

Although service caching has been well studied in cloud networks [3], [4], [5], [6], [7], [10], [11], [12], [15], [18], [21], [24], [27], [32], [34], [37], existing approaches often do not consider server capacity constraints or the quality of service (QoS) requirements of users [3], [6], [7], [10], [12], [26], or do not elaborate the interconnection between the origin service in remote clouds and the service instances at local edge servers [10], [24], [32], [37], or neglected the budget or costs for caching services [3], [10], [11], [15], [18], [21], [27]. To the best of our knowledge, we are the first to consider the problem of service caching in a multi-tiered edge cloud network, with the objective to minimize the delay experienced by users, subject to the capacity constraints of cloudlets and the budget for service caching. We also consider the update communications between origin services at remote data centers and the cached service instances at edge cloudlets, such as instance upgrade or status updates.

The main contributions of this paper are as follows: we formulate a novel *capacitated service caching problem* in a multi-tiered edge cloud network with both identical and different user data packet rates. To solve the problem, we first develop an efficient approximation algorithm with an approximation ratio for the special case where each request demands a single type of service, and the packet rates of the requests are identical. We then extend the algorithm to support the problem with requests having different packet rates. Third, we propose a heuristic algorithm for the problem with each request demanding multiple types of services. We finally evaluate the performance of the proposed algorithms by extensive experiments in a real test-bed, and experimental results demonstrate that our algorithms significantly outperform some existing methods significantly.

The remainder of the paper is arranged as follows. Section 2 introduces the system model and problem formulation. Two approximation algorithms for the capacitated service caching problem will be proposed in Section 3. Section 4 develops an efficient heuristic algorithm to the problem. Section 5 provides experimental evaluation results of the proposed algorithms in a real test-bed. Section 6 summarizes the related work, and Section 7 concludes the paper.

2 Models and Problems

In this section, we first introduce the multi-tiered edge cloud model. We then introduce service caching in the multi-tiered edge cloud network, cost and delay models. We finally formulate the problem formally. The symbols used in this paper are summarized in Table 1.

2.1 Multi-Tiered Edge Cloud Model

We consider a multi-tiered edge cloud network $G = (\mathcal{CL} \cup \mathcal{DC}, E)$ consisting of not only cloudlets that are deployed in a Wireless Metropolitan Area Network (WMAN) for the sake of serving users' requests within their proximity, but also large scale data centers with abundant computing resources in remote areas (Fig. 1). Denote a cloudlet in $\mathcal{CL}$ by CL_i, and let DC_j be a data center in $\mathcal{DC}$. E is a set of links that interconnect the cloudlets and data centers in $\mathcal{CL} \cup \mathcal{DC}$, and let $e \in E$ be such a link in E. Each cloudlet CL_i has a few commodity servers with a quantity of computing resource to implement various delay-sensitive services. Denote the computing capacity of each cloudlet CL_i by $C(CL_i)$. Each data center DC_j has a set of services that can be cached in the cloudlets to improve the QoS of users. We do not consider the capacity constraints of data centers, as they usually have a large number of servers and can provide abundant resources. The data of users' requests are transmitted between users and servers in cloudlets or data centers via links in E. Let c_e and d_e be the cost and delay of transmitting a unit amount of data via link $e \in E$, respectively.

2.2 Service Caching and Users' Requests

In the past decades, with the development of cloud services, various service providers have deployed different services and applications in large-scale data centers to improve service efficiency, reduce running cost, and enhance the flexibility of their business. For example, Cloud Virtual Reality (Cloud VR) incorporates cloud computing and cloud rendering into VR services, cloud-based display output and audio output are coded, compressed, and transmitted to user terminals, implementing cloud-based VR services content and content rendering.[2]

However, such delay-sensitive VR services in large-scale data centers usually suffer from prohibitively long delay, because of the congested core network and high transmission delay between data centers and VR headsets. To decrease such long delays, services that are originally deployed in remote data centers can be *cached* into cloudlets through *service caching*, as shown in Fig. 1. Here the *service caching* means *service placement*, it refers to caching service instances and their related databases or libraries in the edge cloudlets [21], thereby enabling timely service provisioning to users demanding the services.

Without loss of generality, we consider there are Q types of services initially residing in the data centers in $\mathcal{DC}$. Let δ_j be a set of services in a data center $DC_j \in \mathcal{DC}$. Denoted by S_q a type of service, with $1 \leq q \leq Q$, and S_q serves a few requests by processing their data. Since service S_q is usually implemented in a virtual machine (VM) in a data center DC_j, we refer to an implementation of service S_q in a VM as its *instance*, which may be cached into a cloudlet to reduce the delay experienced of the users requiring the service. If an instance of service S_q is cached in a cloudlet, a request of

2. Cloud VR Solution White Paper. https://www.huawei.com/minisite/pdf/ilab/cloud_vr_solutions_wp_en.pdf

TABLE 1
Symbols

Symbols	Meaning
$G = (\mathcal{CL} \cup \mathcal{DC}, E)$	a multi-tiered edge cloud network with a set $\mathcal{CL}$ of cloudlets, a set $\mathcal{DC}$ of data centers, and a set E of links
CL_i, DC_j, e	a cloudlet in $\mathcal{CL}$, a data center in $\mathcal{DC}$, and a link in E, respectively.
c_e, d_e	the cost and delay of transmitting a unit amount of data via link $e \in E$
Q	the types of services residing in the data centers in $\mathcal{DC}$
S_q	a type of service with $1 \le q \le Q$
δ_j	a set of services in a data center $DC_j \in \mathcal{DC}$
R, r_m	a set of requests waiting to be executed in G, and a request in R
ds_m, ρ_m	the data source and the packet rate of request r_m
c_q^{ins}	the cost of instantiating an instance of service S_q
c_m^{trans}, c_m^{up}, c_m^{proc}	the transmission cost, update cost, and processing cost of request r_m
$c_{i,m}$, $c_{j,m}$	the cost of processing a unit amount of data of request r_m in the cloudlet CL_i or data center DC_j
c_m	the cost of implementing a request r_m by caching an instance of its demanded service in a cloudlet in $\mathcal{CL}$
μ_q	packet processing rate for a service instance of S_q
$C(S_q)$	the amount of computing resource to be assigned to an instance of S_q to guarantee its processing rate μ_q
$C(CL_i)$	the computing capacity of cloudlet CL_i
p_{ds_m,CL_i}, p_{ds_m,DC_j}	the path with minimum transmission cost to route the traffic of request r_m from its data source ds_m to the cloudlet CL_i or data center DC_j
$p_{DC_{r_m},CL_{r_m}}$	the path with minimum transmission cost that is used to update the service data regularly
θ	the ratio of packet rate ρ_m to the updating packet rate of request r_m
λ	the ratio of packet rate ρ_m to the size of the processing result
β	a parameter to leverage a trade-off between delay and cost
B	a *caching budget* to limit the total cost of service caching for implementing all user requests
$d_p(r_m)$	the packet processing delay for request r_m with packet rate ρ_m
$d(r_m)$	The delay experienced by request r_m
$d_t(r_m, CL_i)$, $d_t(r_m, DC_j)$	the network delay experienced by request r_m to cloudlet CL_i or data center DC_j
d_{max}, d_{min}	the maximum and minimum delay of implementing a request
c_{max}, c_{min}	the maximum and minimum cost of implementing a request
ρ_{max}, ρ_{min}	the maximum and minimum packer rate of requests in R
α, η, γ	$\alpha = \frac{d_{min}}{d_{max}}$, $\eta = \frac{c_{min}}{c_{max}}$, $\gamma = \arg\max_{r_m \in R} \frac{\rho_m}{\rho_{min}}$
N_{CL}, N_q	the estimated number of cached instances of S_q to be placed in the cloudlets by considering the resource capacity constraint of cloudlets or budget constraint
$\hat{K}$	the estimated number of cached instances for services
OPT	the optimal solution to the capacitated service caching problem
$OPT_{\hat{K}}$	the optimal solution when at most $\hat{K}$ instances will be cached in the cloudlets of the WMAN
CL_{i,κ_i}	a virtual cloudlet for CL_i
R'_m, $r_{m,\omega}$	The set of virtual requests for r_m, and the ωth virtual request of r_m
$d'(r_{m,\omega})$	$d'(r_{m,\omega})$ the delay of implementing $r_{m,\omega}$
$d_{max}(r_{m,\omega})$	the maximum delay experienced by virtual requests of r_m
$d^{v*}(r_{m,\omega})$	the delay experienced by the original request of $r_{m,\omega}$
$d_{\hat{K}}(r_m)$	the delay of r_m in solution $OPT_{\hat{K}}$.
K^*	the maximum number of services that can be cached in the cloudlets in $\mathcal{CL}$ by OPT
R_q	the set of requests that require service S_q

a user who requires the service tends to be assigned to the cloudlet; otherwise, the request will be sent to a data center where the *original instance* of the service is deployed.

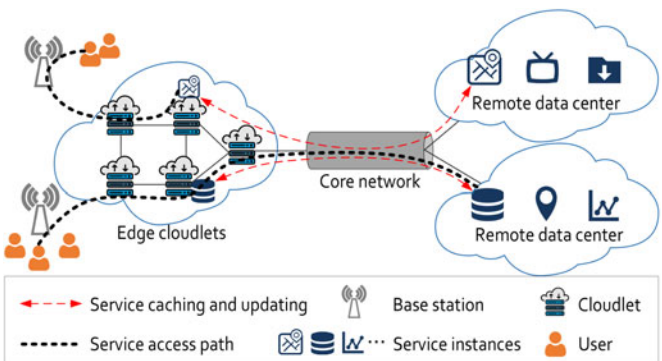


Fig. 1. An example of service caching in a multi-tiered edge cloud network.

Each request with some amount of data is considered as a data flow with a specified data source and demanded service. For a set R of requests waiting to be executed in the multi-tier edge cloud network G, a request $r_m \in R$ can be represented as a tuple $r_m = < ds_m, S_q, \rho_m >$, where ds_m, S_q, and ρ_m are the data source, demanded service, and packet rate, respectively. The *packet rate* ρ_m means the number of generated packets per unit time.

2.3 Cost and Budget Model of Service Providers

Provisioning services to execute users' requests on data centers and caching service instances on cloudlets incurs a non-negligible cost for service providers due to the resource usage and data transmissions. Following existing studies [33], [34], we consider the following types of costs in this paper.

Service Instantiation Cost. Deploying services on data centers and caching instances of the services in cloudlets consume electricity and hardware resources of both data centers and cloudlets. Thus, the service instantiation cost is incurred due to virtual machine (VM) instantiation for a service instance. Let c_q^{ins} be the cost of instantiating an instance of service S_q.

Transmission Cost. For each request r_m, its data and processing results are transmitted between itself and the location (cloudlet or data center) where its requested service S_q is located. If S_q is cached in the cloudlet CL_i, the transmission cost between the data source ds_m of request r_m and CL_i can be calculated by $c_m^{trans} = \sum_{e \in p_{ds_m, CL_i}} c_e \times \rho_m$, where p_{ds_m, CL_i} is the path with minimum transmission cost to route the traffic of request r_m from its data source ds_m to the cloudlet CL_i, c_e is the cost of transmitting a unit amount of data via link e, ρ_m is the packet rate of request r_m. The path may consist of multiple links in G. Similarly, if the request r_m is directed to the service S_q deployed at a data center, the transmission cost is computed by $c_m^{trans} = \sum_{e \in p_{ds_m, DC_j}} c_e \times \rho_m$, where p_{ds_m, DC_j} is the path with minimum transmission cost to route the related data of request r_m from ds_m to the data center DC_j.

Processing Cost. Request executing involves data processing, and data processing charges apply for each unit amount (e.g., Gigabyte) processed by the service regardless of the data source or destination, which will incur processing cost. Let $c_{i,m}$ and $c_{j,m}$ be the cost of processing a unit amount of data of request r_m in the cloudlet CL_i and data center DC_j, respectively. The processing cost by the cached instance of service S_q in the cloudlet CL_i thus is $c_m^{proc,i} = c_{i,m} \times \rho_m$, the processing cost by using service S_q in the data center DC_j thus is $c_m^{proc,j} = c_{j,m} \times \rho_m$. Here, ρ_m is the packet rate of the request r_m.

Update Cost. We consider stateful network services, each service instance will generate and keep state information on the host that serves the requests. The state information needs to be shared and updated with the original service, which is critical for a stateful service [2]. Specifically, when a service instance processes the packets of a request, it will dynamically generate some status data simultaneously. Such status data usually are needed by other cached or original service instance to guarantee the correct processing of data. Considering that the cached instance may not be able to be permanently cached into the multi-tiered edge cloud network due to the resource capacities, its generated status data need to be updated to its original instance for the correct processing of future packets. We thus need to update the data traffic between each original service in data centers and its cached instances in cloudlets. The update cost c_m^{up} of request r_m thus is $c_m^{up} = \sum_{e \in p_{DC_{r_m}, CL_{r_m}}} \theta \times c_e \times \rho_m$, where θ is the ratio of packet rate ρ_m to the updating packet rate of request r_m, and $p_{DC_{r_m}, CL_{r_m}}$ is the path with minimum transmission cost that is used to update the service data regularly.

The cost c_m of implementing a request r_m by caching an instance of its demanded service in a cloudlet in $\mathcal{CL}$ of a multi-tiered edge cloud network G thus is $c_m = c_q^{ins} + c_m^{trans} + c_m^{proc} + c_m^{up}$.

Caching services in cloudlets incurs costs that directly impact the revenue of network service providers. To prevent excessive usage of resources in the cloudlets, a network service provider of the multi-tiered edge cloud network usually has a *caching budget* B to limit the total cost of service caching for all users' requests, i.e.,

$$\sum_{r_m \in R} c_m \leq B, \tag{1}$$

where r_m is a request in R, and c_m is the cost for caching an instance of a type of service demanded by request r_m.

2.4 Delay Model

The delay experienced by each request r_m mainly consists of packet processing delay and network delay for data transmission.

Packet Processing Delay. Each instance of service S_q is assigned with an amount $C(S_q)$ of computing resource to guarantee its packet processing rate μ_q. The packet processing delay $d_p(r_m)$ of service S_q for request r_m with packet rate ρ_m is proportional to the volume of data to process. The packet processing delay $d_p(r_m)$ of r_m thus is,

$$d_p(r_m) = \frac{\rho_m}{\mu_q}. \tag{2}$$

Network Delay. If an instance of service S_q is cached in a cloudlet CL_i, the network delay $d_t(r_m, CL_i)$ experienced by request r_m which is served by the service instance consists of two parts, one part is due to the data transmission from the data source of r_m to the cached service instance at CL_i, and the other part is to transmit the processing results back to the request, which can be calculated by

$$d_t(r_m, CL_i) = \sum_{e \in p_{ds_m, CL_i}} (1 + \lambda) \times d_e \times \rho_m, \tag{3}$$

where λ is the ratio of packet rate ρ_m to the size of the processing result. Otherwise, if no instance of service S_q is cached in a cloudlet, the network delay suffered by request r_m is due to the data transmission between its data source ds_m and data center DC_j with the original instance of S_q, i.e.,

$$d_t(r_m, DC_j) = \sum_{e \in p_{ds_m, DC_j}} (1 + \lambda) \times d_e \times \rho_m. \tag{4}$$

The delay $d(r_m)$ experienced by request r_m thus can be written as

$$d(r_m) = \begin{cases} d_p(r_m) + d_t(r_m, CL_i), & \text{if } S_q \text{ is cached in } CL_i \\ d_p(r_m) + d_t(r_m, DC_j), & \text{otherwise.} \end{cases} \tag{5}$$

2.5 Problem Definition

We consider a multi-tiered edge cloud network $G = (\mathcal{CL} \cup \mathcal{DC}, E)$ with a set δ_j of initially deployed network services in each data center DC_j with $1 \leq j \leq J$, and a set of requests $R = \{r_m \mid 1 \leq m \leq M\}$, assuming that the original instances of the services have enough resource capability to implement all requests in R. It however may incur prohibitive long delay if requests are implemented in remote data centers. We consider the following optimization problems with the aim to shorten the delay experienced by requests.

Assuming that the arrival information of requests in R is given as a priori, and the total resource capacities of the multi-tiered edge cloud network G is larger than the accumulative resource demand of all the requests in R, *the capacitated service caching problem* is to find a set of cloudlets to cache instances of each service $S_q \in \delta_j$ that is originally deployed at each data center DC_j, such that the average delay experienced by the requests is minimized, subject to the computing capacity constraint of each cloudlet $CL_i \in \mathcal{CL}$ and the caching budget constraint B of caching services in the cloudlets.

2.6 NP-Hardness

We show the capacitated service caching problem is NP-hard by a polynomial reduction from another NP-hard problem - the capacitated K-median problem. We are given a set F of facilities where each facility l has a capacity $u_l \in \mathbb{Z}_{>0}$, a set C of clients, a metric d over $F \cup C$, and an upper bound K on the number of facilities we can open, *the capacitated K-median problem* is to find a set $F' \subset F$ of at most K open facilities and a connection assignment $C \longleftarrow F'$ of clients to open facilities, so as to minimize the connection cost [16].

Lemma 1. *The capacitated service caching problem in a multi-tiered edge cloud network $G = (\mathcal{CL} \cup \mathcal{DC}, E)$ is NP-hard.*

Proof. We reduce the capacitated K-median problem to the capacitated service caching problem as follows. Consider the capacitated K-median problem in a given metric complete graph $G = (\mathcal{CL} \cup \mathcal{DC}, E)$. We construct an auxiliary graph $G' = (\mathcal{CL} \cup \mathcal{DC} \cup R, E)$ from G, and the cloudlets in G' have a capacity constraint. There is a set R of requests with each request $r_m \in R$ demanding a type S_q of service, implementing a request r_m by caching an instance of its demanded service S_q in a cloudlet in $\mathcal{CL}$ will consume c_m cost. Assuming caching an instance consumes one unit of the budget B, there are K instances that can be cached if $K = B$. Each instance can be considered as a facility, *the capacitated service caching problem* is to cache K instances in cloudlets such that the average delay experienced by requests is minimized. We can see that an optimal solution to the capacitated service caching problem in G' is also an optimal solution to the capacitated K-median problem in G. Since the capacitated K-median problem is NP-hard [16], the capacitated service caching problem is NP-hard too. □

Notice that the *capacitated service caching problem* differs from the knapsack problem [19]. In knapsack problem, packing an item into a bin only consumes resource from the selected bin, whereas in the capacitated service problem, it consists of not only the service caching and resource consumption, but also scheduling a request to a cached service instance in an edge cloudlet [12], [28], [31].

3 Service Caching With a Single Service Type

We consider a special case of the capacitated service caching problem with a single service type in the data centers in $\mathcal{DC}$. We first propose an approximation algorithm when the requests in R have an identical packet rate ρ, and then extend the algorithm to handle requests with different packet rates. Here, the packet rate refers to the number of packets generated by a request per second (unit time).

3.1 Approximation Algorithm With Identical Packet Rates

Overview and Basic Idea. Given a single type of service S_q whose original instance is initially deployed in the data center DC_j, we propose an approximation algorithm to solve the problem by reducing the capacitated service caching problem with a single service type to the capacitated K-median problem. An approximate solution to the latter will return a feasible solution to the former. For the sake of clarity, we first describe the identical capacitated K-median (UCKM) problem in the following, before elaborating on our basic idea of the approximation algorithm.

In the UCKM problem, we are given a set F of facilities with an identical capacity $u_l \in \mathbb{Z}_{>0}$ for $l \in F$, a set C of clients, a metric d on $F \cup C$, and an upper bound K on the number of facilities we can open. A solution to the UCKM problem is to determine a set $F' \subset F$ of facilities to open, and find an assignment $C \longrightarrow F'$ connecting each client to one of the open facilities so that the number of open facilities is no bigger than K, i.e., $|F'| \leq K$, and the connection cost is minimized while the capacity constraint of each facility is met.

The basic idea of the proposed approximation algorithm is to consider an instance of service S_q as a 'facility' in the UCKM problem, a given number of instances for service S_q are cached into the cloud network. There however exist two difficulties of reducing the capacitated service caching problem into a UCKM problem: (1) Determining how many instances of S_q should be cached into the cloudlets. The number (i.e., K) is given as a priori in the UCKM problem, but the determination of the number in the capacitated service caching problem is difficult, as there are both service caching budget and computing capacity constraints, and (2) a cloudlet can host multiple instances of S_q while a facility in the UCKM problem can only be cached into a single location. How to incorporate such difference is crucial for the reduction of service caching in multi-tiered edge cloud network. Therefore, our proposed algorithm first estimates the number of instances of each service S_q that can be cached, and then caches the estimated number of instances of service S_q by reducing the problem into a UCKM problem. In the following, we thus describe the estimation of instance number for S_q that can be cached.

The Estimation of the Number of Instances of each Type of Service that can be Cached. A network service provider has a caching budget B which specifies the maximum expenditure that can be spent on caching instances of service S_q in cloudlets. On the other hand, in the UCKM problem, only a limited number of facilities can be opened, as there is a budget of opening facilities. We thus use the caching budget B to estimate the number of instances of service S_q that can be cached in the cloudlets.

Recall that the cost of implementing a request in a cached service instance usually is not known in advance, as it depends on the parameters of the request, and the request assignment. The range of the implementation cost however can be estimated from historical statistic information of requests and candidate cache locations. For example, when implementing requests by purchasing EC2 instances in Amazon EC2 On-Demand Pricing, the cost can be vary from \$0.0255 to \$3.9641 per hour.[3] Without loss of generality, we assume that the maximum implementation cost c_{max} and minimum implementation cost c_{min} of a request are known in the multi-tiered edge cloud network G. That is, c_{max} is the implementation cost of a request r_m by a cached service instance in a cloudlet CL_i with the highest instantiation, processing, and transmission cost. c_{min} can be calculated similarly.

A simple method is to use c_{max} or c_{min} to estimate the maximum number of requests that can be implemented without violating the caching budget constraint B. This however can be over pessimistic or optimistic. For example, if a number $\frac{B}{c_{max}}$ of services is estimated to implement the requests in $\mathcal{R}$, this may make most requests be implemented in their original services, thereby increasing the average delay. However, if a number $\frac{B}{c_{min}}$ of services is going to be cached in the cloudlets, the budget B for caching those service instances may be violated. Here we introduce a parameter β to leverage a non-trivial trade-off between the delay experienced by a request and the cost of implementing the requests. Considering the caching budget B, the number of instances of S_q can be estimated by

$$N_q = \Big\lfloor \frac{B}{c_{min} + \beta \times (c_{max} - c_{min})} \Big\rfloor, \tag{6}$$

with the value of β in the range of $[0, 1]$.

Recall that each instance of service S_q should be assigned with $C(S_q)$ computing resource to guarantee its processing rate μ_q. This means that the accumulative packet rates that are processed by service S_q could not exceed its processing rate μ_q to avoid waiting delay. Therefore, the total resource available in the cloudlets of $\mathcal{CL}$ should be greater than $N_q \times C(S_q)$ resources to make sure all the resource demands of the N_q estimated cached instances are met. We thus estimate the number of cached instances of S_q to be cached in the cloudlets due to resource capacity constraint of cloudlets by

$$N_{CL} = \sum_{CL_i \in \mathcal{CL}} \Big\lfloor \frac{C(CL_i)}{C(S_q)} \Big\rfloor. \tag{7}$$

The estimated number $\hat{K}$ of cached instances for S_q thus is

$$\hat{K} = \min\{N_q, N_{CL}\}, \tag{8}$$

depending on whether the caching budget or the capacity constraint of cloudlets is the bottleneck of caching more instances of S_q.

Problem Reduction. We now reduce the capacitated service caching problem with a single type of service S_q to the UCKM problem with maximally K facilities that can be opened.

We first consider $\hat{K}$ as the maximum number of service instances that can be cached in the cloudlets of the multi-tiered edge cloud network G. We then determine the facility set F in the UCKM problem. Recall that in the UCKM problem a facility can be either opened or closed; while each cloudlet can cache multiple instances of S_q as long as its computing resource capacity is not violated. A cloudlet thus cannot be considered as a facility. Instead, each cloudlet CL_i can maximally cache $\Big\lfloor \frac{C(CL_i)}{C(S_q)} \Big\rfloor$ instances of service S_q. We thus create $\lfloor \frac{C(CL_i)}{C(S_q)} \rfloor$ *virtual cloudlets* for CL_i. Denote by CL_{i,κ_i} one of the $\lfloor C(CL_i)/C(S_q) \rfloor$ virtual cloudlets for CL_i. Each virtual cloudlet CL_{i,κ_i} thus is considered as a facility in the UCKM problem. If virtual cloudlet CL_{i,κ_i} is opened to implement requests, there will be a cached instance of S_q in the cloudlet CL_i of the multi-tiered edge cloud network G. The metric between request r_m and virtual cloudlet CL_{i,κ_i} is set as the delay experienced by the request r_m. In addition, the set of data centers having original instances of service S_q can also serve as facilities to implement requests. We say an original instance is 'opened' if it is assigned with some requests to implement; otherwise, we say it is 'closed'.

With the problem reduced to the UCKM problem, we then invoke the approximation algorithm by [1]. Notice that requests may be assigned to different virtual cloudlets. Such requests that are assigned to the virtual cloudlets of cloudlet CL_i will all be assigned to CL_i. The detailed steps of the proposed algorithm are described in Algorithm 1, which is referred to as `Appro_Uni`. An example of `Appro_Uni` is illustrated in Fig. 2.

Algorithm 1. `Appro_Uni`

Input: A multi-tiered edge cloud network G with a set $\mathcal{CL}$ of cloudlets in the WMAN and a set $\mathcal{DC}$ of remote data centers in the core network, a type of service S_q, budget B for caching instances of service S_q at edge cloudlets, and a set of requests R with an identical packet rate ρ that require service S_q.

Output: The number and locations of cached instances for service S_q in the cloudlets, and the assignment of the requests r_m to either the cached instances in the cloudlets or the original instance in the data centers.

1: Estimate the number $\hat{K}$ of cached instances of service S_q that can be cached in the cloudlets by Eq. (8), considering the total computing capacity of cloudlets in $\mathcal{CL}$ and the budget B for caching instances of S_q;

2: Consider $\hat{K}$ service instances as the maximum number of open facilities in the UCKM problem;

3: Split each cloudlet into $\lfloor C(CL_i)/C(S_q) \rfloor$ virtual cloudlets, each of which is considered as a potential location for an open facility in the UCKM problem;

4: Invoke the approximation algorithm by [1] to obtain an approximate solution to the problem;

5: For the cached instances that are assigned to the virtual cloudlets of cloudlet CL_i, cache them into cloudlet CL_i and assign the requests to CL_i;

3.2 Approximation Algorithm With Requests Having Different Packet Rates

We now relax the requests with identical packet rates to the case of requests with different packet rates. Based on `Appro_Uni`, we devise another approximation algorithm for requests

3. https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls

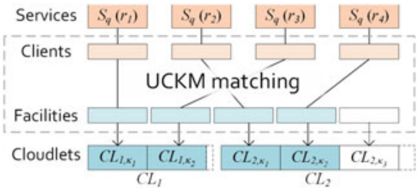


Fig. 2. There are a single type of service S_q whose original instance is initially deployed in the data center DC_j, four requests with identical packet rate (r_1, r_2, r_3, and r_4), and two cloudlets (CL_1 and CL_2). The four requests demanding service S_q correspond to four clients in the UCKM problem. Each instance of service S_q is considered as a "facility" in the UCKM problem. The number $K = 4$ of instances of service S_q is predicted that can be cached in the cloudlets, according to the caching budget B and the total resource capacities of cloudlets CL_1 and CL_2. The UCKM matching generates a matching with minimum delay between requests and cached service instances.

with different packet rates. Specifically, we assume that the minimum and maximum packet rate of requests are given as a priori. Let ρ_{min} and ρ_{max} be the minimum and maximum packet rates of the requests in R, respectively. We assume that ρ_{min} serves as a *basic packet rate*, and the packet rate ρ_m of each request r_m is dividable by ρ_{min}, i.e., $\frac{\rho_m}{\rho_{min}} \in \mathbb{Z}^+$.

The basic idea of the proposed algorithm is to divide each request r_m as a set of *virtual requests* with identical packet rate ρ_{min}. The set of virtual requests for each r_m thus is

$$R'_m = \{r_{m,1}, r_{m,2}, \ldots, r_{m,\omega}, \ldots r_{m,\frac{\rho_m}{\rho_{min}}}\}, \tag{9}$$

where $r_{m,\omega}$ is the ωth virtual request of r_m with $1 \le \omega \le \frac{\rho_m}{\rho_{min}}$. Given the virtual requests with an identical packet rate of ρ_{min}, we then invoke algorithm Appro_Uni. However, the algorithms may assign the virtual requests of a single real request to different cached instances of service S_q. This makes the obtained solution by Appro_Uni not feasible, since the data traffic of each request usually needs to be processed by a single instance of service S_q.

Algorithm 2. Appro_Diff

Input: A multi-tiered edge cloud network G with a set $\mathcal{CL}$ of cloudlets in the WMAN and a set $\mathcal{DC}$ of remote data centers in the core network, a type of service S_q, budget B for caching instances of service S_q at edge cloudlets, a set R of requests that require service S_q with each request r_m having a packet rate ρ_m, the maximum and minimum packet rates of the requests, i.e., ρ_{max} and ρ_{min}.

Output: The number and locations of cached instances for service S_q in the cloudlets, and the assignment of the requests S_q to either the cached instances in the cloudlets or the original instance in the data centers.

1: Split each request r_m into $\frac{\rho_m}{\rho_{min}}$ virtual requests with each having an identical packet rate ρ_{min};
2: Invoke approximation algorithm Appro_Uni to obtain an approximate solution that treats each request as a set of virtual requests, the solution may assign the virtual requests of a single request into different instances of service S_q;
3: Re-assign all the virtual requests of the request r_m to the cached instance of service S_q that incurs a maximum delay for a virtual request;

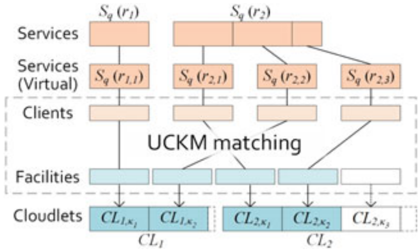


Fig. 3. There are a single type of service S_q whose original instance is initially deployed in the data center DC_j, two requests with different packet rates (r_1 and r_2), and two cloudlets (CL_1 and CL_2). Each request is first split into ρ_m/ρ_{min} virtual requests with each having an identical packet rate. Then similar processing in Fig. 2 is employed in Fig. 3.

We thus need to modify the obtained solution to a feasible solution. Therefore, we adjust the locations for the virtual requests of a request r_m if they are not assigned to a single cached instance of service S_q. Denote by $d'(r_{m,\omega})$ the delay of implementing the ωth virtual request $r_{m,\omega}$. We then assign all the virtual requests to the location (either a cloudlet or a data center) with the maximum delay, i.e., the location that incurs the delay of $\max_{1 \le \omega \le \frac{\rho_m}{\rho_{min}}} d'(r_{m,\omega})$. In this way, the 'split' requests are implemented in a single instance of service S_q.

The details of the algorithm are shown in Algorithm 2, which is referred to as Appro_Diff for simplicity. An example of Appro_Diff is depicted in Fig. 3.

3.3 Algorithm Analysis

We first show the feasibility of the proposed algorithm Appro_Uni in the following lemma.

Lemma 2. *Algorithm Appro_Uni delivers a feasible solution to the capacitated service caching problem.*

Proof. To show the feasibility of the proposed approximation algorithm Appro_Uni, we need to show that (1) all requests are implemented by either cached or original instances of S_q, (2) the violation of the caching budget is constrained by a small factor, and (3) the computing capacity constraint of each cloudlet CL_i is met.

We first show that all requests are implemented by either cached or original instances of S_q. In the first stage of algorithm Appro_Uni, we estimate the number of instances of S_q that can be cached into the cloudlets. According to Eq. (8), we know that not all requests can be implemented by the cached instances of S_q due to the caching budget constraint and the computing resource constraint of cloudlets. In the second stage, we invoke the algorithm for the UCKM problem by considering both the virtual cloudlets of each cloudlet CL_i and the data centers with the original instances of service S_q as potential locations to implement requests. Considering that the original services are already deployed in the data centers, the allocated computing resource is enough to implement all the requests in R. Therefore, we claim that all the requests can be implemented by either cached or original instances of S_q.

We then show that the violation of caching budget B is constrained by a maximum ratio of $\frac{1}{(1-\beta)\times\eta+\beta}$, where $\eta = \frac{c_{min}}{c_{max}}$, and the computing capacity constraint of each cloudlet is met in the following two cases: case (1) $N_q \leq N_{CL}$, and case (2) $N_q > N_{CL}$.

For case (1), it means that the caching budget is the major bottleneck of caching more instances of S_q to minimize the delays of its requests. Since a number N_q of service instances are cached, they can implement at most N_q requests. In the worst case, each request that is implemented by each of the N_q cached service has a maximum implementation cost c_{max}. The total cost of implementing those requests thus is $N_q \times c_{max}$. The violation of the caching budget constraint thus can be calculated by

$$\begin{aligned}\frac{N_q \times c_{max}}{B} &= \Big\lfloor \frac{B}{(c_{min} + \beta(c_{max} - c_{min}))} \Big\rfloor \frac{c_{max}}{B} \\ &\leq \frac{B}{(c_{min} + \beta(c_{max} - c_{min}))} \frac{c_{max}}{B} \\ &\leq \frac{c_{max}}{(c_{min} + \beta(c_{max} - c_{min}))} = \frac{1}{(1-\beta) \times \eta + \beta}.\end{aligned} \tag{10}$$

In case (2), recall that $N_{CL} = \sum_{CL_i \in \mathcal{CL}} \left\lfloor \frac{C(CL_i)}{C(S_q)} \right\rfloor$, the computing resource capacity of cloudlets becomes the major bottleneck of caching more instances of S_q. Due to the fact that the capacity of each facility in the UCKM is met, the computing resource capacity of each cloudlet is met as well. □

Theorem 1. *The approximation ratio of the proposed algorithm* `Appro_Uni` *is* $\frac{|R|}{|R| \times (7+\epsilon) \times \alpha - B(2-\eta) \times (1-\alpha)}$, *if* $c_{min} \geq 1$, *where* ϵ *is an accuracy parameter in the approximation algorithm,* $\alpha = \frac{d_{min}}{d_{max}}$, d_{max} *is the maximum delay of implementing a request in a data center, and* d_{min} *is the minimum delay of implementing the request in a cloudlet. The running time of the algorithm is* $2^{O(\hat{K} \log \hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$.

Proof. We first show the approximation ratio of `Appro_Uni`. Notice that the delay of implementing a request in a data center is far greater than its implementation in a cloudlet. The optimal solution to the capacitated service caching problem thus prefers to cache as many services as possible, such that a maximum number of requests in R are implemented in cloudlets of the WMAN. Let OPT be the optimal solution to the capacitated service caching problem. Denote by K^* the maximum number of services that can be cached in the cloudlets in $\mathcal{CL}$ by OPT, the caching budget constraint B and the computing capacity constraint of cloudlets thus should be met. Considering the caching budget constraint B, it is clear that a maximum number $\lfloor \frac{B}{c_{min}} \rfloor$ of instances can be cached into the WMAN, when all requests can be implemented at the minimum cost c_{min}. We thus can estimate an upper bound of K^* by

$$K^* \leq \min\Big\{ \lfloor \frac{B}{c_{min}} \rfloor, N_{CL} \Big\}. \tag{11}$$

On the other hand, the algorithm `Appro_Uni` estimates that there are at most $\hat{K}$ cached instances for service S_q.

Let $D_{\hat{K}}$ be the average delay experienced by the requests processed by the $\hat{K}$ cached instances of service S_q by algorithm `Appro_Uni`. Let $OPT_{\hat{K}}$ be the optimal solution when at most $\hat{K}$ instances will be cached in the cloudlets of the WMAN. Clearly, we have $OPT \leq OPT_{\hat{K}}$ because $K^* \geq \hat{K}$ and each of the additional request has an improved delay by assigning the request to the original instance in remote data center. Let $d_{\hat{K}}(r_m)$ be the delay of r_m in solution $OPT_{\hat{K}}$. According to the approximation algorithm for the UCKM problem in [1], we have $\frac{D_{\hat{K}}}{OPT_{\hat{K}}} = \frac{D_{\hat{K}}}{\sum_{r_m \in R} d_{\hat{K}}(r_m)} = 7 + \epsilon$.

To analyze the gap between solution OPT and the solution $D_{\hat{K}}$ to the algorithm `Appro_Uni`, we consider the following two cases: (1) the caching budget B is the major bottleneck of caching more instances of S_q, and (2) the computing capacity constraint of cloudlets is the major bottleneck.

For case (1), we have $K^* = O(\lfloor \frac{B}{c_{min}} \rfloor)$ and $\hat{K} = \lfloor \frac{B}{c_{min} + \beta \times (c_{max} - c_{min})} \rfloor$. Clearly, we have $K^* \geq \hat{K}$. There are maximally $(K^* - \hat{K})$ instances that are cached by solution OPT while not cached by solution $D_{\hat{K}}$. In the worst case, all the requests served by the $(K^* - \hat{K})$ cached instances in the optimal solution are served by their original instance in data centers. The $(K^* - \hat{K})$ cached instances can serve at most $(K^* - \hat{K})$ requests, whose upper bound is shown by

$$\begin{aligned}&K^* - \hat{K} \\ &\leq \lfloor \frac{B}{c_{min}} \rfloor - \lfloor \frac{B}{(c_{min} + \beta(c_{max} - c_{min}))} \rfloor \\ &\leq \frac{B}{c_{min}} + 1 - \frac{B}{(c_{min} + \beta(c_{max} - c_{min}))} \\ &\leq \frac{2 \times B}{c_{min}} - \frac{B}{c_{min} + \beta(c_{max} - c_{min})} \leq \frac{2 \times B}{c_{min}} - \frac{B}{c_{max}} \\ &= \frac{B(2c_{max} - c_{min})}{c_{min} c_{max}} = \frac{B(2-\eta)}{c_{min}} \leq B(2-\eta), \text{since } c_{min} \geq 1.\end{aligned} \tag{12}$$

Let R' be the set of requests that are implemented by $(K^* - \hat{K})$ cached service instances in OPT. Let $\alpha = \frac{d_{min}}{d_{max}}$, we then can calculate the approximation ratio by

$$\begin{aligned}\frac{D_{\hat{K}}}{OPT} &= \frac{D_{\hat{K}} \times |R|}{D_{\hat{K}} \times |R| - \sum_{r_m \in R'} (d(r_m) - d^*(r_m))} \\ &\leq \frac{\sum_{r_m \in R} d(r_m)}{\sum_{r_m \in R} d(r_m) - \sum_{r_m \in R'} (d(r_m) - d^*(r_m))} \\ &\leq \frac{|R| \times d_{max}}{|R| \times (7+\epsilon) \times d_{\hat{K}}(r_m) - \sum_{r_m \in R'} (d(r_m) - d^*(r_m))} \\ &\leq \frac{|R| \times d_{max}}{|R| \times (7+\epsilon) \times d_{min} - B(2-\eta)(d_{max} - d_{min})} \\ &= \frac{|R|}{|R| \times (7+\epsilon) \times \alpha - B(2-\eta)(1-\alpha)},\end{aligned} \tag{13}$$

For case (2), when computing capacity is a bottleneck of admitting more requests, as algorithm `Appro_Uni` and OPT can both cache at most N_{CL} instances of S_q, we have $OPT_{\hat{K}} = OPT$. The approximation ratio thus is $7 + \epsilon$.

In summary, the approximation ratio of algorithm Appro_Uni is $\frac{|R|}{|R|\times(7+\epsilon)\times\alpha - B(2-\eta)(1-\alpha)}$.

We then show the running time of the proposed algorithm. It can be seen that algorithm Appro_Uni consists of three phases: the first phase calculates the number of instances of S_q that can be cached, the second stage solves the UCKM problem by treating each cloudlet as $\frac{C(CL_i)}{C(S_q)}$ virtual cloudlets, and the third phase re-assigns requests allocated to virtual cloudlets to their corresponding cloudlets. It is clear that the most time-consuming part is the second phase that takes $2^{O(\hat{K}\log\hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$ time, according to the approximation algorithm in [1]. The theorem holds. □

We now analyze the performance of our proposed algorithm Appro_Diff in the following theorem.

Theorem 2. *There is an approximation algorithm, i.e., algorithm Appro_Diff, for the capacitated service caching problem with a single service type and different packet rates. Its approximation ratio is* $\frac{\gamma|R|}{|R|\times(7+\epsilon)\times\alpha - B(2-\eta)(1-\alpha)}$ *and running time is* $2^{O(\hat{K}\log\hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$, *where* $\eta = \frac{c_{min}}{c_{max}}$, c_{min} *and* c_{max} *are the minimum and maximum implementation costs of all requests,* $\gamma = \arg\max_{r_m\in R}\frac{\rho_m}{\rho_{min}}$.

Proof. We first derive the approximation ratio of algorithm Appro_Diff. Let D^v_{diff} be the average delay experienced by a request in algorithm Appro_Uni through treating each request r_m as $\frac{\rho_m}{\rho_{min}}$ virtual requests. Let D_{diff} be the average delay obtained by algorithm Appro_Diff. We can see from the steps of algorithm Appro_Diff that the difference between D^v_{diff} and D_{diff} is due to the adjustments of virtual requests of a request that are assigned to multiple instances of S_q to a single instance. Specifically, for the request with its $\frac{\rho_m}{\rho_{min}}$ virtual requests being assigned to multiple instances of S_q, all of its virtual requests will be reassigned to the virtual request with the highest delay. Denote by $d_{max}(r_{m,\omega})$ be the maximum delay experienced by virtual requests of r_m. Clearly, we have $\sum_{\omega=1}^{\frac{\rho_m}{\rho_{min}}} d(r_{m,\omega}) \geq d_{max}(r_{m,\omega})$. In addition, the number of virtual requests that are reassigned is no more than $|R'_m|$ $(=\frac{\rho_m}{\rho_{min}})$ of the number of virtual requests with a delay of $d_{max}(r_{m,\omega})$. So $\gamma = \arg\max_{r_m\in R}\frac{\rho_m}{\rho_{min}} = \arg\max_{r_m\in R}|R'_m|$. We then have the following inequality,

$$\begin{aligned} D^v_{diff} &= \frac{\sum_{r_m\in R}\sum_{\omega=1}^{\frac{\rho_m}{\rho_{min}}} d(r_{m,\omega})}{\sum_{r_m\in R}\frac{\rho_m}{\rho_{min}}} \\ &\geq \frac{\sum_{r_m\in R} d_{max}(r_{m,\omega})}{\sum_{r_m\in R}\frac{\rho_m}{\rho_{min}}}, \text{since } \sum_{\omega=1}^{\frac{\rho_m}{\rho_{min}}} d(r_{m,\omega}) \geq d_{max}(r_{m,\omega}) \\ &= \frac{\sum_{r_m\in R} d_{max}(r_{m,\omega})}{\sum_{r_m\in R}|R'_m|} \geq \frac{\sum_{r_m\in R} d_{max}(r_{m,\omega})}{\sum_{r_m\in R}\gamma} \\ &= \frac{\sum_{r_m\in R} d_{max}(r_{m,\omega})}{\gamma\times|R|} = \frac{D_{diff}}{\gamma}. \end{aligned} \tag{14}$$

In other words, we have

$$D_{diff} \leq \gamma \times D^v_{diff}, \tag{15}$$

which means that the average delay obtained by algorithm Appro_Diff will be no greater than γ times the average delay experienced by the solution achieved through treating each request r_m as $\frac{\rho_m}{\rho_{min}}$ virtual requests.

Let D^*_{diff} be the average delay obtained by the optimal solution to the capacitated service caching problem with a single service type. Denote by D^{v*}_{diff} the average delay obtained by the optimal solution of the problem that treating each request r_m as $\frac{\rho_m}{\rho_{min}}$ virtual requests. We now show that D^{v*}_{diff} is no greater than D^*_{diff} as follows.

$$\begin{aligned} D^{v*}_{diff} &= \frac{\sum_{r_m\in R}\sum_{\omega=1}^{\frac{\rho_m}{\rho_{min}}} d^{v*}(r_{m,\omega})}{\sum_{r_m\in R}\frac{\rho_m}{\rho_{min}}} \leq \frac{\sum_{r_m\in R} d^*(r_{m,\omega})}{\sum_{r_m\in R}\frac{\rho_m}{\rho_{min}}} \\ &\leq \frac{\sum_{r_m\in R} d^*(r_{m,\omega})}{|R|}, \text{since } \frac{\rho_m}{\rho_{min}} \geq 1 = D^*_{diff}, \end{aligned} \tag{16}$$

where $d^{v*}(r_{m,\omega})$ is the delay experienced by a request in the solution that treats each request as several virtual ones. Note that the first line in Eq. (16) is that the total delay by all virtual requests is a lower bound of the total delay of all requests, as each request is split into different instances of S_q.

Having the aforementioned inequalities and Theorem 1, we have

$$\begin{aligned} D_{diff} &\leq \gamma D^v_{diff} \\ &\leq \frac{\gamma|R|}{|R|\times(7+\epsilon)\times\alpha - B(2-\eta)(1-\alpha)} D^{v*}_{diff} \\ &\leq \frac{\gamma|R|}{|R|\times(7+\epsilon)\times\alpha - B(2-\eta)(1-\alpha)} D^*_{diff}. \end{aligned} \tag{17}$$

The approximation ratio of algorithm Appro_Diff is $\frac{\gamma|R|}{|R|\times(7+\epsilon)\times\alpha - B(2-\eta)(1-\alpha)}$.

For the running time, note that the running time of the approximation algorithm Appro_Uni is not related to the number of requests. Also, the major difference of algorithms Appro_Uni and Appro_Diff is that algorithm Appro_Diff treats each request as a set of virtual requests. The running time of algorithm Appro_Diff thus is $2^{O(\hat{K}\log\hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$. □

4 Caching With Multiple Service Types

We now consider the capacitated service caching problem with multiple types of services, by proposing an efficient heuristic algorithm for it.

Recall that the objective of the capacitated service caching problem is to minimize the average delay experienced by the requests in R. Intuitively, the more instances are implemented in cloudlets the lower delay the requests will experience. For the Q types of services in the data centers of $\mathcal{DC}$, our proposed algorithm thus should cache the service instances on the cloudlets with the aim to implement more requests. Consequently, services with the following features should be cached first: (1) serving a large number of requests, and (2) processing the packets with small processing rates. Denote by R_q the set of requests that require service S_q, we then rank the Q types of services by allocating weights of the features extracted from R_q, i.e.,

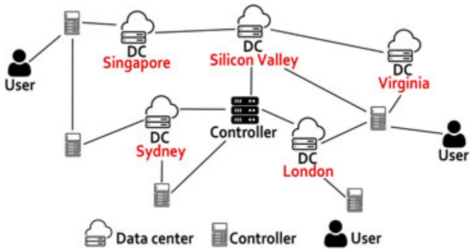


Fig. 4. The topology of the test-bed with leased VMs and a controller.

$$Rank_q = w_1 \times |R_q| + w_2 \times \frac{\sum_{r_m \in R_q} \rho_m}{|R_q|} + w_3 \times \sum_{r_m \in R_q} \rho_m, \tag{18}$$

where w_1, w_2, and w_3 are the weights indicating the importance of the number of requests in R_q that demanding service S_q, the average packet rate of the requests in R_q, and the total packet rate in R_q. $w_1 + w_2 + w_3 = 1$. Notice that to obtain the values of those weights, we run the algorithm multiple times with different network topologies following a reinforcement learning (RL) process. That is, within each run, the agent in the RL process will observe the revenue of increasing or decreasing a small step for w_1, w_2, and w_3, where the revenue is the decrease of the average delay experienced by the requests in R.

The detailed steps are shown in Algorithm 3, which is referred to as `Heuristic` for clarity.

Algorithm 3. `Heuristic`

Input: A multi-tiered edge cloud network G with a set $\mathcal{CL}$ of cloudlets in the WMAN and a set $\mathcal{DC}$ of remote data centers in the core network, multiple types Q of service, budget B for caching instances of service S_q at edge cloudlets, a set R of requests that require services with each request r_m having a packet rate ρ_m, the maximum and minimum packet rates of the requests, i.e., ρ_{max} and ρ_{min}.

Output: The number and the locations of cached instances for each type of service S_q in the cloudlets, and the assignment of the requests S_q to either the cached instances in the cloudlets or the original instance in the data centers.

1: Rank the Q types of services according to the features of (1) the number of requests demanding each type of service S_q, (2) the average packet rate of the requests demanding each type S_q of service, and (3) the total packet rate of the requests demanding S_q, into an increasing order of the values calculated by Eq. (18);

2: **for** each service S_q in the ranked list **do**

3: Invoke algorithm `Appro_Diff` to cache a number of service instances of S_q to cloudlets in $\mathcal{CL}$ and assign the requests demanding S_q to the cached instances or the original instances in data centers in $\mathcal{DC}$;

Theorem 3. *The proposed algorithm* `Heuristic` *delivers a feasible solution to the capacitated service caching problem in a multi-tiered edge cloud network in* $|Q|\log|Q| + |Q| \times 2^{O(\hat{K}\log\hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$ *time.*

Proof. The running time of ranking Q types of services into an increasing order takes $O(|Q|\log|Q|)$. For each type of service S_q, invoking algorithm `Appro_Diff` consumes $2^{O(\hat{K}\log\hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$ time according to Theorem 2. There are Q types of services in total, so for the Q types of services, invoking algorithm `Appro_Diff` consumes $|Q|\times 2^{O(\hat{K}\log\hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$ time. Therefore, the algorithm `Heuristic` consumes $|Q|\log|Q| + |Q| \times 2^{O(\hat{K}\log\hat{K})} \times (|\mathcal{CL}| + |\mathcal{DC}|)^{O(1)}$ time. □

TABLE 2
The Locations and Numbers of Data Centers and Cloudlets

Location	The numbers of data centers	The number of cloudlets
Silicon Valley	1	5
Sydney	1	5
London	1	3
Singapore	1	3
Virginia	1	4
Frankfurt	0	5

5 Experiments

In this section, we evaluate the performance of the proposed algorithms in a real test-bed.

5.1 Experimental Environment

As shown in Fig. 4, we lease a few virtual machines (VMs) with different computing capacities at locations Silicon Valley, Virginia, Sydney, Singapore, London, and Frankfurt from the cloud service provider Alibaba Cloud.[4]

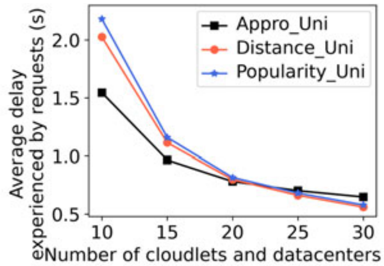
We have one data center and five cloudlets in Silicon Valley and Sydney respectively, one data center and three cloudlets at London and Singapore respectively, one data center and four cloudlets at Virginia, and five cloudlets at Frankfurt. The locations and numbers of data centers and cloudlets are listed in Table 2.

Each data center has five large VMs, and the number of VMs of each cloudlet is withdrawn randomly from the range of $[1, 3]$. Although the scale of each data center or cloudlet in this test-bed can not be comparable to a large-scale data center and or cloudlet, the implementation can be easily extended to a test-bed with large-scale computing nodes (data center or cloudlet). There is also a controller that is responsible for executing the proposed algorithms and implemented in a server with a 24-core Intel Xeon Gold processor and 128 GB memory. The proposed algorithms are implemented as Python programs in the controller. The implementation is open source, and we published it on GitHub.[5]

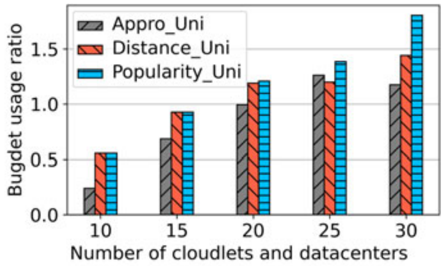
In the experiment, we have five different types of services and 200 requests. Each request produces several Gigabytes of data, we thus emulate the volume of data generated by each request is in the range of [0.1, 1.5] GB. The packet rate of each request is randomly drawn in $[20, 100]$ Mbps.

The rental fee of a cloudlet and a data center is 5 and 80 dollars per month, which means that the rent fee is 0.007 and 0.119 dollars per hour. Each service is instantiated in a Docker container. The instantiation cost of each type of

4. Alibaba Cloud. https://www.alibabacloud.com/.
5. Test-bed implementation. https://github.com/hangvane/serviceCachingEdge



(a) Average delay experienced by requests.



(b) Budget usage ratio.

Fig. 5. The performance of the proposed algorithms against benchmarks with a single service type and identical packet rates on the test-bed.

service depends on the lifetime and size of the container. The processing cost depends on the size of data to be processed. The transmission cost and update cost rely on the flow rate and the transmission time. The budget for caching a service to serve a request is set to 0.3 dollars in average. The trade-off parameter β is set to 0.1. The processing delay and transmission delay are measured based on the real scenario in an online manner.

We evaluate the proposed algorithms against two existing works [11] and [8]. One benchmark [11] adopts a popularity-based heuristic in the caching of services. Its variants with a single service type and multiple service types with identical or different packet rates are referred to as `Popularity_Uni` and `Popularity_Diff`, `Pop_Uni` and `Pop_Diff`, respectively. Another benchmark work [8] first locates the nearest cloudlet to a user who demands a service, if the resource of the cloudlet can satisfy the demand of an instance of the service, and an instance of the service will be cached at the cloudlet. Similarly, we call different variants of the benchmark work as `Distance_Uni`, `Distance_Diff`, `Dis_Uni`, and `Dis_Diff`, respectively.

Unless otherwise specified, we will adopt these default settings in our experiments. Each value in the figures is the mean of the results by applying the mentioned algorithms 15 times on the multi-tiered edge cloud network.

The detailed running process of the test-bed is as follows. First, each node sends a registration request to the controller, to upload its node type, IP, capacity, and other information, and fetch the topology from controller. Having the topology, each node measures the delay with the other nodes and uploads the results to the controller. The controller then generates the request set R according to the experiment parameters described above.

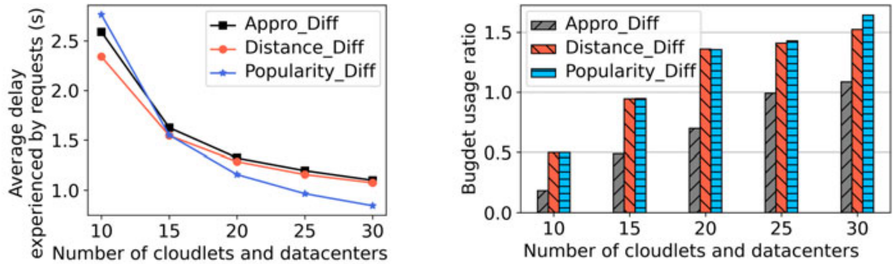
Once the controller obtains all the needed information, it will call a specific algorithm for request assigning. The request set and the assignment are pushed from controller to every user node, and the user nodes start to make requests. If a cloudlet or a data center receives a request from a user, it will instantiate a docker container of Iperf,[6] establish a connection with the user, receive the traffic, and send back some control traffic. The delay experienced by requests thus can be measured from the moment that the instantiation of Iperf sender begins to the moment that the connection is established. We do not generate the computing load of request processing, since the instantiation of Docker container and traffic generation can already consume a significant amount of computing resources (each cloudlet can serve 5 to 15 connections, depending on the request parameters). Once a user obtains the metrics of all the requests assigned to it, it will upload the metrics to the controller. The controller aggregates all the results for analysis and output. We repeat the experiment process, invoke different algorithms, and do the comparison, to show the advancement of the proposed algorithms.

6. Iperf, a network speed test tool that can also be used to generate network traffic. https://iperf.fr/.

5.2 Algorithm Performance Evaluation

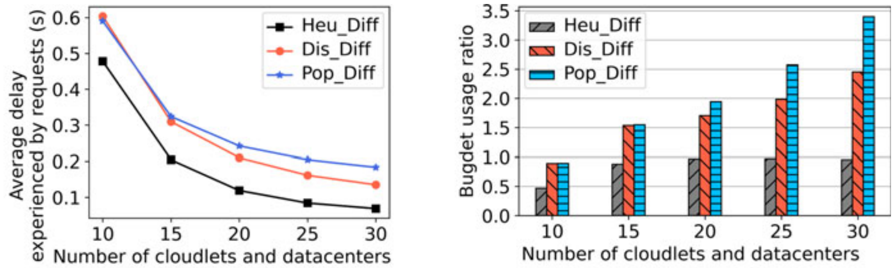
We first evaluate our proposed algorithms against the benchmarks, by varying the number of nodes from 10 to 30. The evaluation results are shown in Figs. 5, 6, 7, and 8. Figs. 5 and 6 shows the case where each request only demands a single type of service. From Fig. 5 we can see that `Appro_Uni` has a delay reduction of up to 29 percent over benchmark `Popularity_Uni` under 5 cloudlets and 5 data centers, while obtains a relatively small budget violation ratio. The gaps between `Appro_Uni` and benchmarks are getting smaller with the growth of node numbers. The rationale is that the proposed algorithms formulate the capacitated service caching problem to the classical UCKM problem, with the considering of the available computing capacity of cloudlets and the caching budget, to obtain an approximate optimal solution with minimum average delay. In contrast, the benchmark based on the popularity of cloudlets only caches service instances at 'hot' and popular cloudlets while ignores the delay factor. Fig. 6 shows that the average delay obtained by benchmark `Popularity_Diff` is up to 23 percent lower than `Appro_Diff`. This is because the slice of requests is not small enough that leads to a waste of cloudlet capacity. Furthermore, the budget constraint limits the caching capability of `Appro_Diff`.

As shown in Figs. 7 and 8, the average delay obtained by the proposed algorithms is 40 and 37 percent lower than those by benchmarks when the number of cloudlets and data centers is 15, while the average cost of the proposed algorithms satisfies the budget constraint. The rationale behind is that the proposed algorithm `Heuristic` adopts a RL-based ranking method to rank the requests by service type, for the objective of caching more service instances on cloudlets to implement requests, so as to minimize the average delay. And benchmarks adopt a greedy strategy that focus on the immediate benefits while ignore the long-term benefits. In addition, the requests cannot be cached when the budget runs out, so the algorithm `Heuristic` with multiple matching iteration has a smaller expected budget violation ratio than that of `Appro_Uni` and `Appro_Diff`.



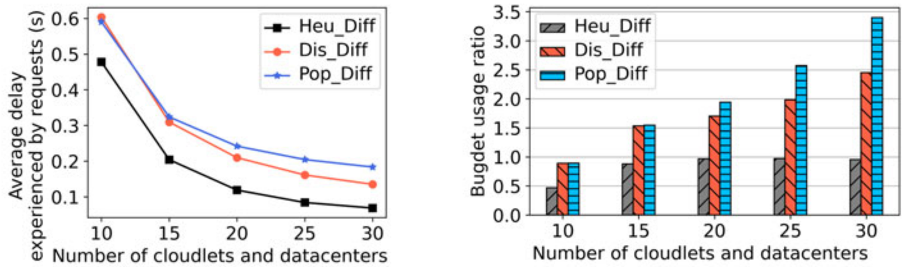
(a) Average delay experienced by requests. (b) Budget usage ratio.

Fig. 6. The performance of the proposed algorithms against benchmarks with a single service type and different packet rates on the test-bed.



(a) Average delay experienced by requests. (b) Budget usage ratio.

Fig. 7. The performance of the proposed algorithms against benchmarks with multiple service types and identical packet rates on the test-bed.



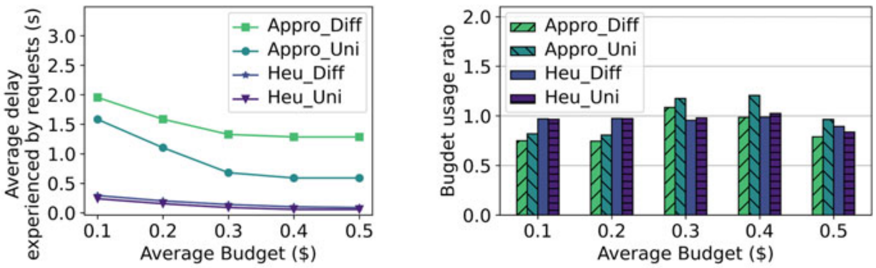
(a) Average delay experienced by requests. (b) Budget usage ratio.

Fig. 8. The performance of the proposed algorithms against benchmarks with multiple service types and different packet rates on the test-bed.

5.3 Impact of Important Parameters on Algorithm Performance

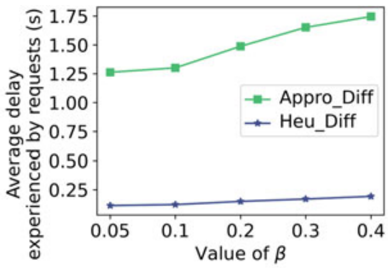
We then investigate the impact of different average budget of requests for caching service instances on the performance of different algorithms under various scenarios on the test-bed, by varying the average budget from 0.1 to 0.5 dollars. The results are shown in Fig. 9, from which we can see that the average delay decreases with the increase of average budget of requests. The rationale is that with the growth of budget for caching instances for a type of service, more service instances can be cached in the cloudlets, and the requirements of requests for accessing the service can be satisfied by instances in their proximity. Besides, the budget violation ratios of `Heu_Uni` and `Heu_Diff` are smaller than that of `Appro_Uni` and `Appro_Diff`, which is consistent with the previous argumentation.

We finally study the impact of parameter β, which leverages a trade-off between the delay experienced by a request and the cost of implementing the requests, by varying the value of β from 0.05 to 0.4. As shown in Fig. 10, with the

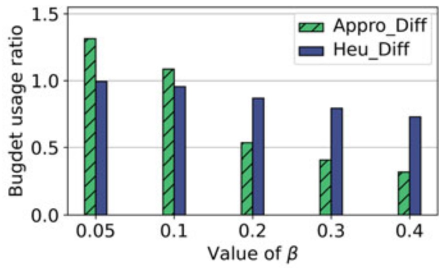


(a) Average delay experienced by requests. (b) Budget usage ratio.

Fig. 9. The impact of the average budget of requests on the performance of the proposed algorithms.



(a) Average delay experienced by requests.



(b) Budget usage ratio.

Fig. 10. The impact of β on the performance of the proposed algorithms.

growth of β, the average delay increases and the budget usage ratio decreases. We notice that when $\beta > 0.1$, the budget usage ratio of `Appro_Diff` is decreasing more significantly than `Heu_Diff`, which means a considerable waste of budget and resources. This is because with multiple matching iteration, not only the expected budget violation ratio, but also the expected budget waste ratio of `Heuristic` is smaller than that of `Appro_Uni` and `Appro_Diff`. We thus suggest setting the value of β to 0.1 in practice.

6 Related Work

There have been extensive related studies focusing on service caching, content caching or service placement in cloud networks [3], [5], [6], [7], [9], [10], [11], [12], [13], [14], [15], [17], [18], [20], [21], [23], [24], [25], [26], [27], [29], [30], [32], [33], [34], [35], [36], [37].

For example, Ascigil *et al.* [3] considered a capacitated service caching problem in edge clouds according to service rankings based on least recently used, the strictest deadline first, etc. Farhadi *et al.* [6] discussed a service placement and request scheduling problem aiming to serve as many requests as possible, with the considering of hardware resources and placement budget. However, they did not explore the various cost of service providers, such as update cost; the QoS requirements of requests in terms of serving delay are not studied neither. Farris *et al.* [7] studied service replication and migration for mobile users to minimize both the quality of experience degradation and the cost of service replica deployment. He *et al.* [12] considered a service provisioning problem with an aim to maximize the number of served requests. They focused on capacity constraints of three types of resources, i.e., communication, computation, and storage, and no placing cost or QoS requirement in terms of delay experienced by users are considered. Xu *et al.* [26] investigated the interactions among content service providers under a novel 'sponsored content' scheme, by proposing a Stackelberg game. The delay and cost issues however are not considered, and their method cannot be directly applied to the service caching problem. However, no QoS requirements or server capacities are considered [3], [6], [7], [9], [10], [12], [26].

Although some works considered capacity constraints of servers or QoS requirements of users, they did not elaborate service update activities between service instance at local servers and original service at data centers [9], [10], [14], [20], [24], [27], [32], [36], [37]. For instance, Ghoreishi [9] formulated virtual caching problems, aims at maximizing the system traffic with budget constraints and caching returns. Jia *et al.* [14] considered a task offloading problem in augmented reality games with an aim to minimize the delay suffered by end-users. Pu *et al.* [20] formulated a joint resource allocation, content placement, and request routing problem in cloud radio access networks. Wang *et al.* [24] presented a problem of provisioning a social VR application on a given set of edge clouds to serve a number of users. They aimed to minimize the total cost for placing services at edge clouds. Xu *et al.* [27] investigated a capacitated service caching problem in MEC-enabled cellular networks, and proposed algorithms based on Lyapunov optimization and Gibbs sampling, aiming to reduce computation delay for users. Yu *et al.* [36] considered an application provisioning and data routing problem, with bandwidth and delay guarantees for data sources, i.e., each data source should receive bandwidth that satisfies its data generation rate, and the transmission delay of each channel should be within the delay tolerance of the application. Xie *et al.* [32] studied the dynamic service caching problem in mobile edge networks with base stations, and develop an efficient algorithm to improve the performance by utilizing the cooperative features of base stations in mobile edge clouds. Zhang *et al.* [37] studied the problem of service placement with an objective to minimize service hosting costs while ensuring critical performance requirements.

Moreover, there are also some studies on service caching that neglect the budget for caching services [3], [10], [11], [13], [15], [18], [21], [27], [36] or did not cover hierarchical framework of the network [13]. For instance, Hou *et al.* [11] studied a content caching problem in mobile networks by predicting content popularity. Jiang *et al.* [13] considered a content caching and delivery problem, for which they placed some popular content items at femto base-stations and user equipments, to minimize the downloading delay of all users, subject to femto base stations' and user equipments' storage capacity and bandwidth capacity of base stations. Li *et al.* [18] investigated a cell caching problem for mobile networks, each cell (base station) can cache popular contents for minimizing delay suffered by users. Xu *et al.* studied problems of service caching among multiple service providers in mobile edge networks [34], the aim is to minimize the total collaboration cost of all network service providers.

Different from the mentioned existing works, we consider a service caching problem in a multi-tiered edge cloud network, we aim to minimize the delay experienced by latencysensitive users, subject to the resource capacity constraints of cloudlets and budget for caching services demanded by the users. In addition, the update activities between original services at remote data centers to the cached service instances at edge cloudlets are elaborated.

7 Conclusion

In this paper, we considered a novel *capacitated service caching problem* in a multi-tiered edge cloud network with both identical and different user data packet rates. We proposed two efficient approximation algorithms with approximation ratios for the problem, based on the identical capacitated K-median (UCKM) algorithm, if requests demand a single type of services. We further devised an efficient and scalable heuristic for the problem with requests demanding multiple types of services. Finally, we conducted extensive experiments on a real test-bed to evaluate the performance of the proposed algorithms against some existing works. Experimental results demonstrate that our proposed algorithms are promising.

Acknowledgments

The authors would like to thank the three anonymous referees and the associate editor for their expertise comments and constructive suggestions, which have helped us improve the quality and presentation of the article greatly. The work of Qiufen Xia and Zichuan Xu was supported by the National Natural Science Foundation of China under Grants 61802047 and 61802048, the General Program of Natural Science Foundation of Liaoning China under Grant 2020–MS–118, and the "Xinghai Scholar Program" in Dalian University of Technology, China.

References

[1] M. Adamczyk, J. Byrka, J. Marcinkowski, S. M. Meesum, and M. Wlodarczyk, "Constant factor FPT approximation for capacitated k-median," 2018, *arXiv:1809.05791*.

[2] K. Amemiya and A. Nakao, "Layer-integrated edge distributed data store for real-time and stateful services," in *Proc. IEEE/IFIP Netw. Oper. Manageme. Symp.*, 2020, pp. 1–9.

[3] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou, "On uncoordinated service placement in edge-clouds," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2017, pp. 41–48.

[4] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 377–390, Feb. 2021.

[5] J. Famaey, W. D. Cock, T. Wauters, B. Dhoedt, and P. Demeester, "A latency-aware algorithm for dynamic service placement in large-scale overlays," in *Proc. IFIP, Int. Symp. Integr. Netw. Manage.*, 2009, pp. 414–421.

[6] V. Farhadi *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," in *Proc. INFOCOM, IEEE Conf. Comput. Commun.*, 2019, pp. 1279–1287.

[7] I. Farris, T. Taleb, M. Bagaa, and H. Flick, "Optimizing service replication for mobile delay-sensitive applications in 5G edge network," in *Proc. Int. Conf. Commun.*, 2017, pp. 1–6.

[8] I. Farris, T. Taleb, A. Iera, and H. Flick, "Lightweight service replication for ultra-short latency applications in mobile edge networks," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–6.

[9] S. E. Ghoreishi, D. Karamshuk, V. Friderikos, N. Sastry, M. Dohler, and A. H. Aghvami, "A cost-driven approach to caching-as-a-service in cloud-based 5G mobile networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 997–1009, May 2020.

[10] B. Gao, Z. Zhou, F. Liu, F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *Proc. INFOCOM, IEEE Conf. Comput. Commun.*, 2019, pp. 1459–1467.

[11] T. Hou, G. Feng, S. Qin, and W. Jiang, "Proactive content caching by exploiting transfer learning for mobile edge computing," in *Proc. IEEE Global Commun. Conf.*, 2017, pp. 1–6.

[12] T. He, H. Khamfroush, S. Wang, T. L. 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. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 365–375.

[13] W. Jiang, G. Feng, and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1382–1393, May 2017.

[14] M. Jia, and W. Liang, "Delay-sensitive multiplayer augmented reality game planning in mobile edge computing," in *Proc. ACM Int. Conf. Model., Anal. Simulation Wireless Mobile Syst.*, 2018, pp. 147–154.

[15] J. Kwak, Y. Kim, L. B. Le, and S. Chong, "Hybrid content caching in 5g wireless networks: Cloud versus edge caching," *IEEE Trans. Wireless Commun.*, vol. 17, no. 5, pp. 3030–3045, May 2018.

[16] S. Li, "Approximating capacitated k-median with $(1 + \epsilon)$k open facilities," in *Proc. Symp. Discrete Algorithms*, 2016, pp. 786–796.

[17] X. Li, X. Wang, K. Li, and V. C. M. Leung, "CaaS: Caching as a service for 5G networks," *IEEE Access*, vol. 5, pp. 5982–5993, 2017.

[18] X. Li, X. Wang, S. Xiao, and V. C. M. Leung, "Delay performance analysis of cooperative cell caching in future mobile networks," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 5652–5657.

[19] D. Pisinger, "Algorithms for Knapsack Problems," Ph.D. thesis, Dept., Comput. Sci. Univ. Copenhagen, Copenhagen, Denmark, 1995.

[20] 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 J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1751–1767,Aug. 2018.

[21] 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 *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 10–18.

[22] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.

[23] B. Varghese, C. Reaño, and F. Silla, "Accelerator virtualization in fog computing: Moving from the cloud to the edge," *IEEE Cloud Comput.*, vol. 5, no. 6, pp. 28–37, Nov./Dec. 2018.

[24] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 468–476.

[25] Q. Wang, Q. Xie, N. Yu, H. Huang, X. Jia, "Dynamic server switching for energy efficient mobile edge networks," *in Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.

[26] X. Xiong, *et al.*, "Joint sponsored and edge caching content service market: A game-theoretic approach," *IEEE Trans. Wireless Commun.*, vol. 18, no. 2, pp. 1166–1181, Feb. 2019.

[27] 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.

[28] Q. Xia, W. Liang, and W. Xu, "Throughput maximization for online request admissions in mobile cloudlets," *IEEE Conf. Local Comput. Netw.*, 2013, pp. 589–596.

[29] Q. Xia, W. Liang, and Z. Xu, "The operational cost minimization in distributed clouds via community-aware user data placements of social networks," *Comput. Netw.*, vol. 112, pp. 263–278,2017.

[30] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, 2016.

[31] Q. Xia, W. Liang, Z. Xu, and B. Zhou, "Online algorithms for location-aware task offloading in multi-tiered mobile clouds," *in Proc. Int. Conf. Utility Cloud Comput.*, 2014, pp. 109–116.

[32] Q. Xie, Q. Wang, N. Yu, H. Huang, and X. Jia, "Dynamic service caching in mobile edge networks," in *Proc. Int. Conf. Mobile Ad Hoc Sens. Syst.*, 2018, pp. 73–79.

[33] Q. Xia, Z. Xu, W. Liang, and A. Y. Zomaya, "Collaboration- and fairness-aware big data management in distributed clouds. *Trans. Parallel Distrib. Syst.*, vol. 27, no. 7, pp.1941–1953, Jul. 2016.

[34] Z. Xu, L. Zhou, S. Chau, W. Liang, Q. Xia, and P. Zhou, "Collaborate or separate? distributed service caching in mobile edge clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2066–2075.

[35] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *Trans. Comput.*, vol. 65, no. 5, pp. 1440–1452, May 2016.

[36] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 783–791.

[37] Q. Zhang *et al.*, "Dynamic service placement in geographically distributed clouds," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 762–772, 2013.

Qiufen Xia (Member, IEEE) received the PhD degree in computer science from the Australian National University in 2017, the ME and BSc degrees in computer science from the Dalian University of Technology in China in 2012 and 2009. She is currently an associate professor with the Dalian University of Technology. Her research interests include edge computing, cloud computing, query evaluations, and big data analytics.

Wenhao Ren received the BSc degree in computer science and technology from Harbin Engineering University in China in 2019. He is currently working toward the PhD degree with the Dalian University of Technology. His research interests include mobile cloud computing, big data processing, and network function virtualization.

Zichuan Xu (Member, IEEE) received the PhD degree in computer science from the Australian National University in 2016, the ME and BSc degrees from the Dalian University of Technology in China in 2011 and 2008. From 2016 to 2017, he was a research associate with Department of Electronic and Electrical Engineering, University College London, U.K. He is currently an associate professor with School of Software, Dalian University of Technology. His research interests include mobile edge computing, network function virtualization, softwaredefined networking, Internet of Things, algorithmic game theory, and optimization problems.

Xin Wang (Senior Member, IEEE) received the PhD degree in electrical and computer engineering from Columbia University, New York. She is currently an associate professor with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, New York. Before joining Stony Brook, she was a member of technical staff in the area of mobile and wireless networking with the Bell Labs Research, Lucent Technologies, New Jersey, and an assistant professor with the Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, New York. Her research interests include algorithm and protocol design in wireless networks and communications, mobile and distributed computing, and networked sensing and detection. She was on the executive committee and technical committee of numerous conferences and funding review panels, and an associate editor for the *IEEE Transactions on Mobile Computing*. She was the recipient of the NSF Career Award in 2005 and ONR Challenge Award in 2010.

Weifa Liang (Senior Member, IEEE) received the BSc degree in computer science from Wuhan University, China, in 1984, the ME degree in computer science from the University of Science and Technology of China in 1989, and the PhD degree in computer science from the Australian National University in 1998. He is currently a professor with the Department of Computer Science, City University of Hong Kong. Prior to joining City University of Hong Kong, he was a professor with the Research School of Computer Science, the Australian National University. He has authored or coauthored papers in top venues. His research interests include wireless sensor networks, mobile edge computing , network function virtulization, Internet of Things, design and analysis of approximation algorithms and online algorithms, parallel and distributed algorithms, combinatorial optimization, and graph theory. His research has been supported by ARC Discovery Grant, Australia Research Council. He was an associate editor for the *IEEE Transactions on Communication.*