Cooperative Service Caching and Workload Scheduling in Mobile Edge Computing

Xiao Ma*, Ao Zhou*, Shan Zhang†, Shangguang Wang*
* State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
†School of Computer Science and Engineering, Beihang University

***Abstract*—Mobile edge computing is beneficial for reducing service response time and core network traffic by pushing cloud functionalities to network edge. Equipped with storage and computation capacities, edge nodes can cache services of resource-intensive and delay-sensitive mobile applications and process the corresponding computation tasks without outsourcing to central clouds. However, the heterogeneity of edge resource capacities and mismatch of edge storage and computation capacities make it difficult to fully utilize both the storage and computation capacities in the absence of edge cooperation. To address this issue, we consider cooperation among edge nodes and investigate cooperative service caching and workload scheduling in mobile edge computing. This problem can be formulated as a mixed integer nonlinear programming problem, which has non-polynomial computation complexity. Addressing this problem faces challenges of subproblem coupling, computation-communication tradeoff, and edge node heterogeneity. We develop an iterative algorithm named ICE to solve this problem. It is designed based on Gibbs sampling, which has provably near-optimal performance, and the idea of water filling, which has polynomial computation complexity. Simulation results demonstrate that our algorithm can jointly reduce the service response time and the outsourcing traffic, compared with the benchmark algorithms.**

***Index Terms*—edge service caching, workload scheduling, mobile edge computing**

I. Introduction

The proliferation of mobile devices and the advancement of Internet of things bring resource-intensive and delay-sensitive mobile applications to the public, such as objective recognition, augmented reality, and mobile gaming. Mobile cloud computing proposes to offload these applications to central clouds, which, however, suffers from unpredictable wide area network delay, making it hard to guarantee the quality of service for delay-sensitive applications [1]–[3]. Moreover, according to Cisco, the growth rate of mobile data required to be processed will far exceed the capacity of central clouds in 2021 [4]. As a result, the outsourcing data traffic and computation load to central clouds become critical concerns of network operators. Mobile edge computing has emerged as a promising solution to addressing above concerns [5], [6]. A typical form of mobile edge computing is to equip mobile base stations with storage and computation capacities (forming edge nodes). Through caching services (including program codes and related databases) of mobile applications at edge nodes, mobile edge computing is able to process the corresponding computation tasks at network edge, benefiting from reduced service response time and outsourcing traffic to central clouds.

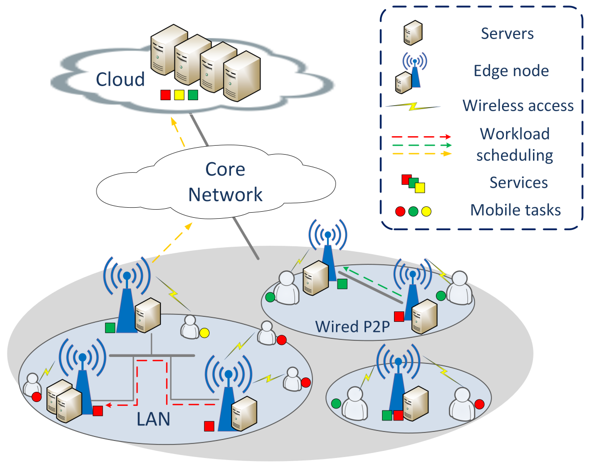


Fig. 1: Cooperative service caching and workload scheduling in mobile edge computing.

Compared with mobile cloud computing which has elastic resource capacity, the main limitation of mobile edge computing is the limited resource capacities of edge nodes. When there is no cooperation among edge nodes, the edge resource capacities are prone to be under-utilized for two reasons. First, the heterogeneity of edge resource capacities can cause resource under-utilization. For an edge node that has insufficient storage capacity to cache a service or cannot provide sufficient computation capacity for an application, the corresponding computation tasks have to be outsourced to central clouds rather than to nearby powerful edge nodes, resulting in under-utilization of edge resources [7]. Moreover, the mismatch of storage and computation capacities of edge nodes further aggravates edge resource wasting. An edge node with large computation capacity cannot process substantial computation tasks when it has insufficient storage capacity to cache the services, leading to under-utilization of edge computation capacities. To fully utilize both the storage and computation capacities of edge nodes, it is crucial to explore the potential of cooperation among edge nodes .

In this paper, we consider cooperation among edge nodes and investigate cooperative service caching and workload scheduling in mobile edge computing. As shown in Fig. 1, neighbouring edge nodes are connected by local area network (LAN) or wired peer-to-peer links [8]. For an edge node that is not caching a service or does not provide sufficient computation capacity, the corresponding computation tasks can be offloaded to nearby under-utilized edge nodes that have cached the service or outsourced to the cloud. Through exploiting the cooperation among edge nodes, the heterogeneous edge resource capacities can be fully utilized and the resource capacity mismatch of individual edge nodes can be alleviated. The existing work which considers edge cooperation and jointly optimizes service caching and workload scheduling has sought to maximize the overall requests served at edge nodes while ensuring the service caching cost within the budget [9], [10]. However, it is hard to determine the exact value of the budget in practical scenarios. Furthermore, while the reduced delay is the main advantage of mobile edge computing, the service response time is not considered as a performance criteria in the existing work. In this paper, we investigate the cooperative service caching and workload scheduling with the objective of minimizing the service response time and the outsourcing traffic (denoted as problem 1).

Solving this problem is challenging. First, service caching and workload scheduling are two coupled subproblems. The service caching policies determine the decision space of workload scheduling, and in return, the workload scheduling results reflect the performance of the service caching policies. Solving problem 1 should take into consideration the interplay between the two subproblems. Second, minimizing the service response time requires to properly trade off the computation and the transmission delay. While offloading computation tasks from overloaded edge nodes to nearby under-utilized edge nodes is beneficial for reducing the computation delay, task offloading causes additional transmission delay on LAN. Solving problem 1 needs to balance the tradeoff between computation and communication. Third, edge nodes are heterogeneous in both storage and computation capacities. Solving problem 1 needs to balance workloads among these heterogeneous edge nodes, causing high computation complexity. How to deal with edge heterogeneity and design algorithms with reduced computation complexity is challenging.

To deal with the challenge of subproblem coupling, we formulate problem 1 as a mixed integer nonlinear programming problem to jointly optimize service caching and workload scheduling. A two-layer Iterative Caching updatE (ICE) algorithm is designed to direct the interplay of the two subproblems, with the outer layer updating the edge caching policies iteratively based on Gibbs sampling (the service caching subproblem) and the inner layer optimizing the workload scheduling polices (the workload scheduling subproblem). To properly trade off the computation and communication delay, we use queuing models to analyze the delay in each part of the system and thereby compute the average service response time. A proper computation-communication tradeoff can be achieved when the average service response time is minimized. To deal with high computation complexity of workload scheduling caused by edge heterogeneity, we exploit the convexity of the workload scheduling subproblem and propose a heuristic workload scheduling algorithm with polynomial computation complexity based on the idea of water filling.

The contributions of this paper are summarized as follows:

- We investigate cooperative service caching and workload scheduling in mobile edge computing, aiming at minimizing both the service response time and outsourcing traffic to central clouds. We formulate this problem as a mixed integer non-linear programming problem and show its non-polynomial complexity by analyzing the simplified cases of this problem.
- We use queuing models to analyze the delay in each part of the system, based on which the convexity of the workload scheduling subproblem is proved.
- We design the two-layer ICE algorithm to solve problem 1. In the outer layer, the proposed ICE algorithm updates the service caching policies iteratively based on Gibbs sampling. In the inner layer, we exploit the convexity of the workload scheduling subproblem and further propose a heuristic workload scheduling algorithm with reduced complexity based on the idea of water-filling.

The reminder of this paper is organized as follows. Section II reviews the related work. Section III analyzes the system model and provides problem formulation. In Section IV, algorithm design is presented in detail, and Section V illustrates simulation results. Finally, the concluding remarks are given in Section VI.

II. Related Work

Mobile edge computing has been envisioned as a promising computing paradigm with the benefits of reduced delay and outsourcing traffic to central clouds. Due to the limited storage and computation capacities of edge nodes, properly placing services of mobile applications and scheduling computation tasks among edge nodes are crucial to optimize the quality of services with high resource efficiency. There has been extensive work devoted to workload scheduling, service caching, and joint service caching and workload scheduling.

Although mobile edge computing enables mobile users to access powerful resources within one-hop range [11], [12], a lot of prior work has evolved to allow task offloading to edge nodes (or the remote cloud) within more than one hop and solve the workload scheduling problem. Online workload scheduling among edge-clouds has been studied in [13], [14] to accommodate dynamic requests in mobile edge computing. In [15], Tong *et al.* have developed a hierarchical architecture of edge cloud servers and optimized workload placement in this architecture. Cui *et al.* have proposed the software defined control over request scheduling among the cooperative mobile cloudlets [16]. All the above work on workload scheduling has a common assumption that each edge node (also named as edge-cloud or cloudlet) has cached all the services and can process any type of computation tasks, which is impractical due to the limited storage capacities of edge nodes. Service caching among edge nodes should also be taken into consideration.

Caching services at edge nodes is an effective approach to relieving the burden of backhual network and central clouds. Increasing efforts have been devoted to edge service caching.

Borst *et al.* [17] have presented popularity-based distributed caching algorithms for content distribution network (CDN). Zhang *et al.* [18] have proposed a user-centric edge caching mechanism in which each user is served by multiple edge servers cooperatively to optimize service delay. Yang *et al.* [19] have designed location-aware edge caching schemes to maximize local hit rates by predicting popularity of contents. Dynamic edge service caching has been extensively studied in [20]–[23]. Dán *et al.* in [20] have investigated prediction-based content placement and provided approximations of dynamic content allocation for the hybrid system of cloud-based storage and CDN. History-based dynamic edge caching have been proposed in [21] without predicting future requests or adopting stochastic models. In mobile edge computing, both storage and computation capacities of edge nodes are limited. To guarantee quality of service for applications that are resource-intensive and delay-sensitive (e.g., augmented reality applications), service caching and workload scheduling should be jointly optimized to minimize service response time with high resource utilization.

Joint optimization of edge caching and request routing has been studied in [24], [25]. The work [24] has been oriented to data-intensive applications (such as video streaming), which, however, cannot be directly applied to applications that are both data-intensive and computation-intensive (such as augmented reality applications). The work [25] has studied the issue in multi-cell environment, which is only applied to users covered by multiple base stations concurrently. To address the above issue, joint optimization of service caching and workload scheduling has been investigated in [7], [9], [10]. The work [7] has jointly optimized service caching and task offloading without considering cooperation among edge nodes, which can lead to under-utilization of heterogeneous edge resource capacities. The work [9] and [10] have investigated joint service caching and request scheduling without taking the service response time (including the transmission delay and the computation delay) as the performance criteria, which cannot highlight the benefit of reduced delay in mobile edge computing. Different from the existing work, we study cooperative service caching and workload scheduling in mobile edge computing, aiming at minimizing the service response time and the outsourcing traffic to central clouds. We solve this problem by developing the iterative caching update algorithm based on Gibbs sampling and further proposing the heuristic workload scheduling algorithm with polynomial complexity based on the idea of water filling.

III. System Model and Problem Formulation

A. System Model

In this paper, we investigate cooperative service caching and workload scheduling in mobile edge computing. As shown in Fig. 1, neighbouring edge nodes are connected by LAN or wired peer-to-peer links. For an edge node that is not caching a service or does not provide sufficient computation capacity, the corresponding computation tasks can be offloaded to the neighbouring under-utilized edge nodes that have cached the service or outsourced to the cloud. We consider a multi-edge system consisting of a set of $\mathbb{N} = \{1, 2, ..., N\}$ edge nodes, each of which is equipped with the computation capacity R_n ($n \in \mathbb{N}$) and storage capacity P_n ($n \in \mathbb{N}$). The system provides a library of $\mathbb{S} = \{1, 2, ...S\}$ services, such as mobile gaming, object recognition, video streaming, etc, which are differentiated by the computation and storage requirements. To process one type of mobile application at network edge, an edge node should provision certain storage capacity to cache the service of the application. Let p_s be the required storage capacity to cache service s. For each service s, we consider that the computation requests of the corresponding computation task (in CPU cycles) follow exponential distribution with the expectation of β_s, and the task arrival at each edge node n is a Poisson process with the expected rate A_{ns}, which is a general assumption [7]. There is a centralized cloud with much larger storage and computation capacity than each edge node. The cloud stores all the services and the processing delay in the cloud d_{cloud} is mainly caused by the transmission delay from edge nodes to the cloud.

1) Edge Caching and Workload Scheduling Policies: We answer two questions in this study: 1) which edge nodes cache each type of service? and 2) how to schedule the computation workloads among the neighbouring edge nodes that have cached the same services? We use two sets of variables to model the edge caching and workload scheduling results: c_{ns} indicates whether service s is cached at edge node n, and λ_{ns} represents the workload ratio of service s that is executed at edge node n. We refer by *edge caching* and *workload scheduling* policies to the respective matrices:

$$\begin{aligned} \boldsymbol{C} &= (c_{ns} \in \{0,1\} : n \in \mathbb{N}, s \in \mathbb{S}), \\ \boldsymbol{\Lambda} &= (\lambda_{ns} \in [0,1] : n \in \mathbb{N} \cup \{o\}, s \in \mathbb{S}). \end{aligned} \tag{1}$$

Here, λ_{os} denotes the workload ratio of service s outsourced to the cloud. Denote by $\boldsymbol{c_n} = (c_{ns} : s \in \mathbb{S})$ the caching decision of edge node n, and $\boldsymbol{C_n}$ the feasible region of $\boldsymbol{c_n}$, i.e., $\boldsymbol{c_n} \in \boldsymbol{C_n}$. The services cached at each edge node n cannot exceed the storage capacity, i.e.,

$$\sum_{s \in \mathbb{S}} c_{ns} p_s \leq P_n. \tag{2}$$

For each service s, the workload ratio λ_{ns} ($n \in \mathbb{N} \cup \{o\}$) satisfies the normalization condition as

$$\sum_{n \in \mathbb{N} \cup \{o\}} \lambda_{ns} = 1. \tag{3}$$

2) Service Response Time: Denote by Θ_n the set of neighbouring edge nodes that have direct connection with edge node n, and d_n the transmission delay on LAN to edge node n. The computation workload executed at edge node n should be less than or equal to the overall arriving tasks of neighbouring edge nodes,

$$\lambda_{ns} A_s \leq \sum_{i \in \Theta_n \cup \{n\}} A_{is}, \tag{4}$$

where A_s is the overall computation workload of service s in the system, i.e. $A_s = \sum_{n \in N} A_{ns}$. We can notice that if

$\lambda_{ns}A_s \leq A_{ns}$, all the tasks are from edge node n; Otherwise, the excessive tasks ($\lambda_{ns}A_s - A_{ns}$) are from nearby edge nodes.

At each edge node, the computation capacity is shared by the cached services. Let the function Γ_n represent the computation allocation mechanism at edge node n, i.e., the computation capacity allocated to service s is $r_{ns} = \Gamma_n(\boldsymbol{C})$. For each service s, as the computation requests of the corresponding computation task follow exponential distribution, the serving time at edge node n also follows exponential distribution with the expectation $\frac{\beta_s}{r_{ns}}$. Moreover, the task arrival of service s at edge node n is a Poisson process with the expectation $\lambda_{ns}A_s$. Thus for each service s, the serving process of computation tasks at edge node n can be modeled as an M/M/1 queue, and the computation delay is

$$D_{ns} = \frac{1}{\mu_{ns} - \lambda_{ns}A_s}, \tag{5}$$

where $\mu_{ns} = \frac{r_{ns}}{\beta_s}$. To ensure the stability of the queue, we have

$$\lambda_{ns}A_s < \mu_{ns}. \tag{6}$$

By combining Eq. (4) and Eq. (6), λ_{ns} is constrained as

$$\lambda_{ns}A_s \leq \min\{\sum_{i\in\Theta_n\cup\{n\}} A_{is}, \mu_{ns} - \varepsilon\}. \tag{7}$$

Here, ε ($\varepsilon > 0$) is introduced to merge Eq. (6) into Eq. (7).

Outsourcing tasks to the cloud suffers from long transmission delay in the core network. As the computation capacity of the cloud is much larger than each edge node, we consider that the computation delay in the cloud is negligible. Thus, the processing delay in the cloud is mainly caused by the transmission delay in the core network. The arrival process of computation tasks in the core network can be modeled as a Poisson process with expected rate $\lambda_{os}A_s$ (which is a sum of multiple independent Poisson processes from edge nodes). Let t_s be the amount of transmission requests (e.g., input data) when outsourcing one unit of computation requests for service s (in CPU cycle). Here, t_s is a constant related to the specific service s [8], [16]. Then for the service s, the transmission requests of a corresponding task follow exponential distribution with expectation $t_s\beta_s$. The transmitting time of a task in the core network also follows exponential distribution with expectation $\frac{t_s\beta_s}{B_s}$, where B_s represents the core network bandwidth for transmitting service s. Hence, the transmitting delay in the core network is given as

$$d_{\text{cloud}} = \frac{1}{\frac{B_s}{t_s\beta_s} - \lambda_{os}A_s}, \tag{8}$$

where $\lambda_{os}A_s < \frac{B_s}{t_s\beta_s}$.

The average response time of service s can be computed as a weighted sum of delay at each part of the system, including the computation delay at edge nodes, the transmission delay within LAN and the transmission delay to the cloud, i.e.,

$$D_s = \sum_{n\in\mathbb{N}} \left[\lambda_{ns}D_{ns} + \frac{\max\{\lambda_{ns}A_s - A_{ns}, 0\}}{A_s} d_n\right] + \lambda_{os}d_{\text{cloud}}. \tag{9}$$

Here, $\frac{\max\{\lambda_{ns}A_s - A_{ns},0\}}{A_s}$ represents the ratio of the workload offloaded to edge node n from neighbouring edge nodes, and d_n is the transmission delay within LAN to edge node n.

B. Problem Formulation

This paper jointly optimizes the edge service caching and workload scheduling policies, aiming at minimizing service response time and overall outsourcing traffic to the cloud:

$$\begin{aligned}
\textbf{P1}: &\min_{\mathbf{C},\mathbf{\Lambda}} \sum_{s\in\mathbb{S}} (D_s + w_s\lambda_{os}A_s) \\
&s.t. \sum_{n\in N\cup\{o\}} \lambda_{ns} = 1, && \forall s \in \mathbb{S} \\
&\mathrm{C1}: \sum_{\mathrm{s}\in\mathrm{S}} \mathrm{c_{ns}p_s} \leq \mathrm{P_n}, && \forall \mathrm{n} \in \mathbb{N} \\
&\mathrm{C2}: \lambda_{\mathrm{ns}}\mathrm{A_s} \leq \min\{\sum_{\mathrm{i}\in\Theta_\mathrm{n}\cup\{\mathrm{n}\}} \mathrm{A_{is}}, \mu_{\mathrm{ns}} - \varepsilon\}, \\
& && \forall n \in \mathbb{N}, \forall s \in \mathbb{S} \\
&\mathrm{C3}: \lambda_{\mathrm{os}}\mathrm{A_s} \leq \frac{\mathrm{B_s}}{\mathrm{t_s}\beta_\mathrm{s}} - \varepsilon, && \forall \mathrm{s} \in \mathbb{S} \\
&\mathrm{C4}: \lambda_{\mathrm{ns}} \geq 0, && \forall \mathrm{n} \in \mathbb{N}\cup\{\mathrm{o}\}, \forall \mathrm{s} \in \mathbb{S} \\
&\mathrm{C5}: \mathrm{c_{ns}} \in \{0, 1\}. && \forall \mathrm{n} \in \mathbb{N}, \forall \mathrm{s} \in \mathbb{S}
\end{aligned} \tag{10}$$

Here w_s is a weight constant which is positively related to the transmitted data traffic when outsourcing tasks of service s. Constraint C1 ensures the cached services at each edge node do not exceed the storage capacity. C2 is the combined result of Eq. (4) and Eq. (6), ensuring that each edge node only admits computation requests from nearby edge nodes, and the computation workload scheduled to each edge node does not exceed the computation capacity for each service.

C. Complexity Analysis

Problem **P1** is a mixed integer nonlinear programming problem. In this section, we present the non-polynomial computation complexity of **P1** by analyzing two simplified cases, i.e., *non-cooperation among edge nodes*, and *considering a single type of service*.

1) Simplified Case 1: Non-cooperation among Edge Nodes: In this case, we assume that there is no cooperation among edge nodes. The computation tasks of different services are either processed locally or directly outsourced to the cloud. Thus, the computation tasks outsourced to the cloud are not only decided by the edge computation capacity, but also highly dependent on the storage capacity of each individual edge node. In this scenario, problem **P1** is reduced to the *service caching and task oursourcing* problem similar to [7]. Specifically, workload scheduling among edge nodes in **P1** is reduced to N independent task outsourcing subproblems. Each edge node only needs to decide the oursourced computation requests $\lambda_{o_n s}$ (which is given as $\lambda_{o_n s} = 1 - \lambda_{ns}$) according to its own service caching policy and the computation capacity limitation. It is indicated in [7] that the reduced *service caching and task outsourcing* problem remains challenging since it is a mixed integer nonlinear programming problem and has non-polynomial computation complexity.

2) Simplified Case 2: Considering a Single Type of Service: In this case, we assume that only a single type of service is considered in the system. With this assumption, the caching result at each edge node can be directly determined by the relationship between the service storage requirement and the edge storage capacity: The service will be cached at one edge node if it has sufficient storage capacity; Otherwise, the service will not be cached. Thus, problem **P1** is reduced to a *workload scheduling* problem, which schedules computation workloads among the edge nodes that have cached the service.

Solving the *workload scheduling* problem is challenging in two folds. First, edge nodes are heterogeneous in both computation task arrivals and edge computation capacities. Balancing the workloads among the heterogeneous edge nodes is critical to minimizing the service response time and the outsourcing traffic to central clouds, which, however, can cause high computation complexity when achieved in a centralized manner. Second, scheduling workloads among edge nodes should consider the computation-communication tradeoff. Offloading computation tasks from overloaded edge nodes to nearby light-loaded edge nodes or to the cloud is beneficial for reducing the computation delay, but meanwhile causes additional transmission delay. Minimizing service response time requires proper balance between the computation and communication delay.

By summarizing the above two simplified cases of problem **P1**, both the reduced *service caching and task outsourcing* and *workload scheduling* problems have non-polynomial computation complexity. Therefore, problem **P1** also has non-polynomial computation complexity.

IV. Algorithm Design

As clarified in the above section, even the simplified cases of problem **P1** remain to be intractable at polynomial complexity. This section presents the main idea of algorithm design which jointly optimizes service caching and workload scheduling policies with reduced computation complexity. Specifically, we design a two-layer Iterative Caching updatE algorithm (ICE). The outer layer updates the edge caching policy based on Gibbs sampling (Algorithm 1) [26]. In the inner layer, the edge caching policy is given and problem **P1** is reduced to the workload scheduling subproblem among the edge nodes that have cached a certain type of service (similar to Simplified case 2). We demonstrate exponential complexity of the workload scheduling subproblem by convexity analysis and further propose a heuristic workload scheduling algorithm (Algorithm 2) with reduced computation complexity based on the idea of water filling.

A. Caching Update based on Gibbs Sampling

In the outer layer of ICE, the edge caching policy is updated based on Gibbs sampling. Gibbs sampling is a Monte Carlo Markov Chain technique, which can deduce the joint distribution of several variables from the conditional distribution samples. The main idea of Gibbs sampling is to simulate the conditional samples by sweeping through each variable while maintaining the rest variables unchanged in each iteration [26]. The Monte Carlo Markov Chain theory guarantees that the stationary distribution deduced from Gibbs sampling is the target joint distribution [27]. In this work, we exploit the idea of Gibbs sampling to determine the optimal service caching policies iteratively, as shown in Algorithm 1. The key point of this algorithm is to associate the conditional probability distribution of edge caching policies with the objective of **P1** (Step 7). Through properly designing the conditional probability in each iteration, the deduced stationary joint distribution can converge to the optimal edge caching policies with high probability.

Algorithm 1 works as follows. In each iteration, randomly select an edge node n ($n \in \mathbb{N}$) and a feasible edge caching decision $\boldsymbol{c}_{\boldsymbol{n}}^*$ while maintaining the caching decisions of the rest edge nodes unchanged (Step 3). With the given caching policies of all the edge nodes, **P1** is reduced to the workload scheduling subproblem:

$$\begin{aligned}
&\mathbf{P2}: \min_{\boldsymbol{\Lambda}} \sum_{s\in\mathbb{S}} (D_s + w_s\lambda_{os}A_s) \\
&s.t. \sum_{n\in N\cup\{o\}} \lambda_{ns} = 1, && \forall s\in\mathbb{S} \\
&\qquad \lambda_{ns}A_s \le \min\{\sum_{i\in\Theta_n\cup\{n\}} A_{is}, \mu_{ns}-\varepsilon\}, && \forall n\in\mathbb{N}, \forall s\in\mathbb{S} \\
&\qquad \lambda_{os}A_s \le \frac{B_s}{t_s\beta_s} - \varepsilon, && \forall s\in\mathbb{S} \\
&\qquad \lambda_{ns} \ge 0. && \forall n\in\mathbb{N}\cup\{o\}, \forall s\in\mathbb{S}
\end{aligned} \tag{11}$$

After solving **P2** optimally, we can obtain the optimal objective value y (defined as $y = \min\limits_{\boldsymbol{\Lambda}} \sum\limits_{s\in\mathbb{S}} (D_s + w_s\lambda_{os})$). Assume that when the selected edge node n changes its caching decision from $\boldsymbol{c_n}$ to $\boldsymbol{c}_{\boldsymbol{n}}^*$, the optimal objective value varies from y to y^*. Then, we associate the conditional probability distribution of edge caching policies with the objective value as: the selected edge node n changes its caching decision from $\boldsymbol{c_n}$ to $\boldsymbol{c}_{\boldsymbol{n}}^*$ with probability $\rho = \frac{1}{1+e^{(y^*-y)/\omega}}$ ($\omega > 0$) and maintains the current caching decision $\boldsymbol{c_n}$ with probability $1-\rho$ (Step 7). Finally, the iteration ends if the stop criterion is satisfied.

The following theorem demonstrates the convergence property of Algorithm 1.

Theorem 1. *Algorithm 1 can converge to the globally optimal solution of problem **P1** with an increasing probability as ω decreases. When $\omega \to 0$, the algorithm converges to the globally optimal solution with probability 1.*

Proof. Please refer to Appendix A. □

Remark: Theorem 1 indicates that in each iteration of the Gibbs sampling technique, through proper selection of ω in $\rho = \frac{1}{1+e^{(y^*-y)/\omega}}$ ($\omega > 0$) which associates the service caching update process with the objective value, the proposed ICE

Algorithm 1 Caching Update Based on Gibbs Sampling

```
Input:
    $A_{ns}$ ($n \in \mathbb{N}, s \in \mathbb{S}$), $p_s$, $\beta_s$ ($s \in \mathbb{S}$))
Output:
    The edge caching policy $\boldsymbol{C}$ and the workload scheduling policy $\boldsymbol{\Lambda}$.
1: Initialize $\boldsymbol{C}^0 \leftarrow \mathbf{0}$.
2: for iteration $i = 1, 2, ...$ do
3:   Randomly select an edge node $n \in \mathbb{N}$ and an edge caching decision $\boldsymbol{c}_n^* \in \boldsymbol{C}_n$.
4:   if $\boldsymbol{c}_n^*$ is feasible then
5:     Based on the edge caching policy $(\boldsymbol{c}_1^{i-1}, .., \boldsymbol{c}_n^{i-1}, .., \boldsymbol{c}_N^{i-1})$, compute the optimal workload scheduling policy $\boldsymbol{\Lambda}$ and the corresponding $y$ by solving P2.
6:     Based on the edge caching policy $(\boldsymbol{c}_1^{i-1}, ..., \boldsymbol{c}_n^{*}, ..., \boldsymbol{c}_N^{i-1})$, compute the optimal workload scheduling policy $\boldsymbol{\Lambda}^*$ and the corresponding $y^*$ by solving P2.
7:     Let $\boldsymbol{c}_n^i = \boldsymbol{c}_n^*$ with the probability $\rho = \frac{1}{1+e^{(y^*-y)/\omega}}$, and $\boldsymbol{c}_n^i = \boldsymbol{c}_n^{i-1}$ with the probability $1-\rho$.
8:   end if
9:   if the stopping criterion is satisfied then
10:    End the iteration and return $\boldsymbol{C}^i$, $\boldsymbol{\lambda}^i$.
11:  end if
12: end for
```

algorithm can converge to the optimal edge caching policy with high probability.

B. Heuristic Workload Scheduling Algorithm

In the inner layer of ICE, the edge caching policy is given and problem **P1** is reduced to the workload scheduling subproblem (problem **P2**) among the edge nodes that have cached a certain type of service. Problem **P2** should be solved to obtain the optimal workload scheduling policy and the corresponding objective value. We first demonstrate the exponential complexity of problem **P2** through theoretical analysis and further propose a heuristic workload scheduling algorithm by exploiting the convexity of the problem.

*1) Computation Complexity of **P2**:* Substitute Eq. (5) and (9) into **P2**, and the objective function $f(\boldsymbol{\Lambda})$ can be rewritten as

$$\begin{aligned} f(\boldsymbol{\Lambda}) =& \sum_{s\in S}(D_s + w_s\lambda_{os}A_s) \\ =& \sum_{s\in S}\sum_{n\in N}\left(\frac{\lambda_{ns}}{\mu_{ns}-\lambda_{ns}A_s} + \frac{\max\{\lambda_{ns}A_s - A_{ns}, 0\}}{A_s}d_n\right) \\ &+ \sum_{s\in S}\left(\frac{\lambda_{os}}{\frac{B_s}{t_s\beta_s} - \lambda_{os}A_s} + w_s\lambda_{os}A_s\right). \end{aligned} \tag{12}$$

Theorem 2. *Problem **P2** is a convex optimization problem over the workload scheduling policy* $\boldsymbol{\Lambda}$.

Proof. Please refer to Appendix B. □

A convex optimization problem can be solved using Karush-Kuhn-Tucker (KKT) conditions [28]. We first present the KKT conditions of **P2**. When the caching policy is given, the computation resources allocated to each service are determined according to $\Gamma_n(\boldsymbol{C})$. Thus for one service, the workload scheduling policy among edge nodes that have cached the service is independent of the other services. Solving problem **P2** is equivalent to optimizing the workload scheduling policy for each type of service. For service s, define the Lagrange function as

$$\begin{aligned} &L_s(\boldsymbol{\lambda_s}, \boldsymbol{\alpha_s}, \boldsymbol{\eta_s}) \\ =&(D_s + w_s\lambda_{os}) + \sum_{n\in\mathbb{N}\cup\{o\}}\alpha_{ns}(\lambda_{ns}A_s - \pi_{ns}) \\ &- \sum_{n\in\mathbb{N}\cup\{o\}}\alpha_{(N+1+n)s}\lambda_{ns} + \eta_s\Big(\sum_{n\in\mathbb{N}\cup\{o\}}\lambda_{ns} - 1\Big), \end{aligned} \tag{13}$$

where $\boldsymbol{\alpha_s}$ and $\boldsymbol{\eta_s}$ are Lagrange multipliers, and π_{ns} is the upper bound of the inequation constraints defined as $\pi_{ns} = \min\{\sum_{i\in\Theta_n\cup\{n\}} A_{is}, \mu_{ns} - \varepsilon\}$ ($n \in \mathbb{N}$) and $\pi_{os} = \frac{B_s}{t_s\beta_s} - \varepsilon$. The KKT conditions of service s are given as

$$\begin{aligned} &(\text{C1})\ \frac{\partial L_s(\boldsymbol{\lambda_s}, \boldsymbol{\alpha_s}, \boldsymbol{\eta_s})}{\partial\lambda_{ns}} = 0, && \forall n \in \mathbb{N}\cup\{o\} \\ &(\text{C2})\ 0 \le \lambda_{ns}A_s \le \pi_{ns}, && \forall n \in \mathbb{N}\cup\{o\} \\ &(\text{C3})\ \sum_{n\in N\cup\{o\}}\lambda_{ns} = 1, && \\ &(\text{C4})\ \alpha_{ns}(\lambda_{ns}A_s - \pi_{ns}) = 0, && \forall n \in \mathbb{N}\cup\{o\} \\ &(\text{C5})\ \alpha_{(N+1+n)s}\lambda_{ns} = 0, && \forall n \in \mathbb{N}\cup\{o\} \\ &(\text{C6})\ \alpha_{ns} \ge 0. && \forall n \in \{1, 2..., 2N+2\} \end{aligned} \tag{14}$$

Here, (C4), (C5) and (C6) arise from the inequation constraints of **P2**. For each inequation constraint, there are two possible results in Eq. (14): 1) $\alpha_{ns} = 0$, $\lambda_{ns}A_s < \pi_{ns}$ (or $\lambda_{ns} > 0$), indicating that the optimal results are at the extreme points derived from (C1); 2) $\alpha_{ns} > 0$,$\lambda_{ns}A_s = \pi_{ns}$ (or $\lambda_{ns} = 0$), indicating the optimal results are at the boundary. As there are $2(N+1)$ inequation constraints in problem **P2** (i.e., the computation capacity constraints of edge nodes), directly searching for the results satisfying the KKT conditions can cause $O(2^{2(N+1)})$ computation complexity. To solve **P2** with reduced complexity, we propose the heuristic workload scheduling algorithm (Algorithm 2).

2) Algorithm Design: The main idea of the algorithm is to first remove the computation capacity constraints of edge nodes and the transmission bandwidth constraint of the core network (i.e., the inequation constraints in **P2**) to derive the correlation of workload scheduling results of edge nodes and the cloud. Then we search for the optimal results satisfying the KKT conditions within the resource constraints.

When removing the inequation constraints, the KKT conditions only keep (C1) and (C3), with (C1) changed to

$$\frac{\partial L_s(\boldsymbol{\lambda_s}, \boldsymbol{\eta_s})}{\partial\lambda_{ns}} = \frac{\partial\Big(D_s + w_s\lambda_{os} + \eta_s\big(\sum_{n\in\mathbb{N}\cup\{o\}}\lambda_{ns} - 1\big)\Big)}{\partial\lambda_{ns}} = 0, \tag{15}$$

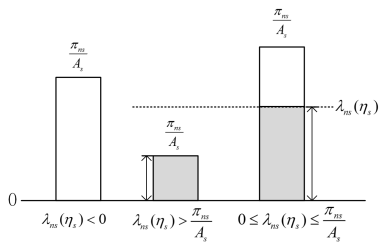

$\frac{\pi_{ns}}{A_s}$
$\frac{\pi_{ns}}{A_s}$
$\frac{\pi_{ns}}{A_s}$
$\lambda_{ns}(\eta_s)$
0
$\lambda_{ns}(\eta_s) < 0$
$\lambda_{ns}(\eta_s) > \frac{\pi_{ns}}{A_s}$
$0 \le \lambda_{ns}(\eta_s) \le \frac{\pi_{ns}}{A_s}$

Fig. 2: Water-filling based workload scheduling.

for all $n \in \mathbb{N} \cup \{o\}$. However, $L_s(\boldsymbol{\lambda_s}, \boldsymbol{\eta_s})$ is not partially derivable over λ_{ns} when $\lambda_{ns} = \frac{A_{ns}}{A_s}$ ($n \in \mathbb{N}$), which is caused by $\frac{\max\{\lambda_{ns}A_s - A_{ns}, 0\}}{A_s}$ in (9). We solve this problem by dividing it into two cases: $\lambda_{ns} < \frac{A_{ns}}{A_s}$ and $\lambda_{ns} \geq \frac{A_{ns}}{A_s}$, and $\lambda_{ns}(\eta_s)$ ($n \in \mathbb{N}$) can be derived as

$$\lambda_{ns} = \begin{cases} \dfrac{1}{A_s}(\mu_{ns} - \sqrt{-\dfrac{\mu_{ns}}{\eta_s + d_n}}), & \text{if } \eta_s \leq -\theta_{ns} - d_n \\ \dfrac{1}{A_s}(\mu_{ns} - \sqrt{-\dfrac{\mu_{ns}}{\eta_s}}), & \text{if } \eta_s \geq -\theta_{ns} \\ \dfrac{A_{ns}}{A_s}, & \text{otherwise} \end{cases} \tag{16}$$

where $\theta_{ns} = \frac{\mu_{ns}}{(\mu_{ns} - A_{ns})^2}$, and λ_{os} is given as

$$\lambda_{os} = \frac{1}{A_s}(\frac{B_s}{t_s\beta_s} - \sqrt{-\frac{B_s}{t_s\beta_s(w_s + \eta_s)}}). \tag{17}$$

After removing the inequation constraints, the workload scheduling policy $\boldsymbol{\Lambda}$ is given as the functions of η_s (Eq. (16), (17)) to satisfy (C1) in the KKT conditions. To obtain the optimal solution of η_s which satisfies the equation constraint and the inequation constraints in **P2**, we search for the workload scheduled to n ($n \in \mathbb{N} \cup \{o\}$) based on the idea of water filling. As shown in Fig. 2, scheduling workloads to edge nodes (or the cloud) is similar to filling water to tubes. When the water level is above the upper bound or beneath the lower bound of the tube, the water cannot be increased or decreased anymore. By combining Eq. (16), (17) with (C2), we have the following conclusion: Let $y(\eta_s) = \sum_{n \in N \cup \{o\}} \lambda_{ns}(\eta_s) - 1$, then $y(\eta_s)$ is constant or monotone decreasing with η_s. Thus, we can search the optimal η_s by the bisection method with the details summarized in Algorithm 2. In Algorithm 2, λ_{ns} ($n \in \mathbb{N} \cup \{o\}$) is traversed to compute $y(\eta_s)$ (step 2-13). From step 16 to 23, the optimal η_s is computed in an iterative manner, with overall $O(\log(\frac{\eta_s^r - \eta_s^l}{\xi}))$ iterations. Therefore, the computation complexity of Algorithm 2 is $O(N \log(\frac{\eta_s^r - \eta_s^l}{\xi}))$, which is linearly increasing with N.

V. Simulation Results

In this section, extensive simulations are conducted to evaluate our algorithm. We simulate a 100m × 100m area covered with 12 edge nodes which provides a total of 8 services. The edge nodes are empowered by heterogeneous storage and computation capacities, both of which follow uniform distribution. The total arrival rates of computation tasks at different edge nodes A_n ($n \in \mathbb{N}$) are uniformly distributed. At each edge node, the popularity of services follow Zipf's distribution, i.e., $\chi_{ns} \propto r_s^{-\upsilon}$, where r_s is the rank of service s and υ is the skewness parameter [10]. Thus, the arrival rate of computation tasks of service s at edge node n can be computed as $A_{ns} = \chi_{ns} \cdot A_n$, where A_n is the total arrival rate of computation tasks at edge node n. The main simulation parameters are listed in Table I.

We compare ICE with two benchmark algorithms.

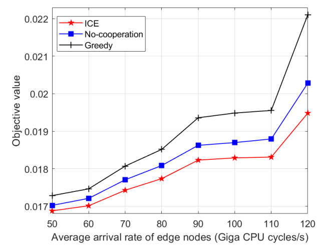
Non-cooperation algorithm [8]: Edge nodes cache services according to Gibbs Sampling. At each edge node, the computation workloads of a service are either processed locally or outsourced to the cloud.

Algorithm 2 Heuristic Workload Scheduling Algorithm

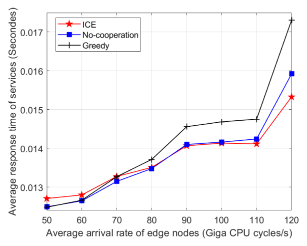
```
1:  Define $y(\eta_s)$ as follows.
2:  for each $n \in \mathbb{N} \cup \{o\}$ do
3:      Compute $\lambda_{ns}(\eta_s)$ according to Eq. (16) and Eq. (17).
4:      if $\lambda_{ns}(\eta_s) < 0$ then
5:          $\lambda_{ns} = 0$.
6:      else
7:          if $\lambda_{ns}(\eta_s) > \frac{\pi_{ns}}{A_s}$ then
8:              $\lambda_{ns} = \frac{\pi_{ns}}{A_s}$.
9:          else
10:             $\lambda_{ns} = \lambda_{ns}(\eta_s)$.
11:         end if
12:     end if
13: end for
14: $y(\eta_s) = \sum_{n \in N \cup \{o\}} \lambda_{ns} - 1$.
15: Find $\eta_s^{\mathrm{l}} < \eta_s^{\mathrm{r}} < 0$ satisfying $y(\eta_s^{\mathrm{l}}) > 0,\ \eta_s^{\mathrm{r}} < 0$.
16: while $\eta_s^{\mathrm{r}} - \eta_s^{\mathrm{l}} \geq \xi$ do
17:     $\eta_s^{\mathrm{m}} = \frac{\eta_s^{\mathrm{l}} + \eta_s^{\mathrm{r}}}{2}$.
18:     if $y(\eta_s^{\mathrm{l}}) \cdot y(\eta_s^{\mathrm{m}}) < 0$ then
19:         $\eta_s^{\mathrm{r}} = \eta_s^{\mathrm{m}}$.
20:     else
21:         $\eta_s^{\mathrm{l}} = \eta_s^{\mathrm{m}}$.
22:     end if
23: end while
24: The optimal result of $\eta_s$ is $\eta_s^* = \frac{\eta_s^{\mathrm{l}} + \eta_s^{\mathrm{r}}}{2}$.
25: Repeat step 2-13 to compute the optimal workload
    scheduling policy $\boldsymbol{\Lambda}^*$.
```

TABLE I: Simulation Parameters

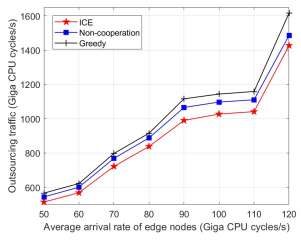
Parameter	Value
Service storage requirement, p_s	[20, 80] GB
Service computation requirement, β_s	[0.1, 0.5] Giga CPU cycles/task
Edge node storage capacity, P_n	[100, 200] GB
Edge node computation capacity, R_n	[50, 100] Giga CPU cycles
Data transmission ratio of service, t_s	[0.1,1.0] Mb/GHz
Core network bandwidth for service, B_s	160 Mbps
Skewness parameter, υ	0.5
Smooth parameter, ω	10^{-6}



(a) Objective value y



(b) Average response time



(c) Average outsourcing traffic

Fig. 3: Performance comparison of different algorithms: the smooth parameter $\omega = 10^{-6}$ and the weight factor $w_s = 6 \cdot 10^{-4}$.

Greedy algorithm: Edge nodes cache services according to popularity. Popular services have high priority to be cached. For the cached services, each edge node optimizes the workloads processed locally and outsourced to the cloud to minimize service response time and outsourcing traffic.

A. Performance Comparison

We compare the three algorithms in terms of the objective value (weighted sum of service response time and outsourcing traffic), service response time and outsourcing traffic by varying the average arrival rate of tasks at edge nodes (i.e., average A_n), and the results are shown in Fig. 3.

Compared with the Non-cooperation algorithm and the Greedy algorithm, our ICE algorithm always yields the minimum object value and outsourcing traffic, and close to minimum total service response time. In the Greedy algorithm, all the edge nodes cache the popular services with high priority, thus the computation tasks of less popular services have to be outsourced to the cloud. Moreover, the Greedy algorithm only relies on service popularity to determine the edge caching policy without considering storage requirements of services. Caching multiple less popular services with low storage requirements at edge nodes can be more beneficial for fully utilizing both the computation and the storage capacities compared with caching one popular service with large storage requirement. The ICE and Non-cooperation algorithms cache services based on Gibbs sampling, taking both the storage requirements of services and service popularity into consideration. Therefore, the Greedy algorithm generally induces more outsourcing traffic and service response time than the other two algorithms. The Non-cooperation algorithm cannot fully utilize the computation capacities of edge nodes which

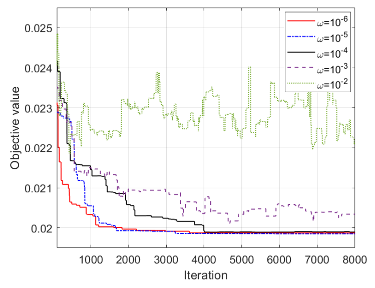


Fig. 4: Impact of ω on the convergence of ICE: the weight factor $w_s = 6 \cdot 10^{-4}$.

have low storage capacities due to the absence of cooperation among edge nodes. In the ICE algorithm, both the storage and computation capacities of edge nodes can be coordinated and fully utilized through careful design of service caching and workload scheduling among the connected edge nodes.

B. Convergence of ICE

According to the theoretical analysis in Theorem 1, the Gibbs sampling based service caching algorithm (Algorithm 1) can converge to the optimal service caching results with probability 1 when the smooth parameter ω is close to 0. This part illustrates the influence of ω on the convergence of ICE with the results shown in Fig. 4. The objective value can converge to the near-optimal results when $\omega \leq 10^{-4}$, and the converging rate is faster as ω decreases. When $\omega \geq 10^{-3}$, the objective value converges slowly to higher value ($\omega = 10^{-3}$) or even cannot converge ($\omega = 10^{-2}$). These results can be explained by Step 7 of ICE and Eq.(22). According to ICE, the smaller is ω, the more probable that the selected edge node updates to the better caching decision in each iteration. Thus, when ω is small, the objective value converges quickly (within less iterations). In addition, it can be concluded from Eq.(22) that stationary probability of the optimal caching result increases with ω, and the probability $\rightarrow 1$ when $\omega \rightarrow 0$. Therefore, the smaller is ω, the more probable that ICE converges to the optimal caching result.

C. The Impact of Edge Node Connectivity

This part analyzes the impact of edge node connectivity on the performance of ICE. As shown in Fig. 5, the system with all the edge nodes connected converges to the minimum objective value while the system with no edge nodes connected has the highest objective value. In the system with all the edge nodes connected, the benefits of cooperation can be achieved at system level through scheduling workloads among all the edge nodes. When edge nodes are partially connected, the cooperation benefits can only be explored within clusters (the edge nodes within a cluster are connected and different clusters are not connected with each other). Therefore, the higher extent that edge nodes are connected with each other, the more cooperation benefits can be achieved by ICE.

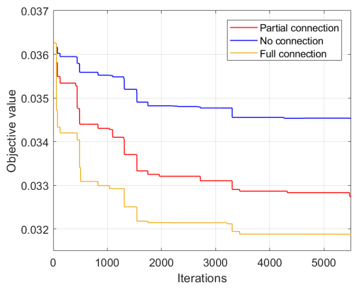


Fig. 5: Impact of edge node connectivity: the smooth factor $\omega = 10^{-6}$ and the weight factor $w_s = 3 \cdot 10^{-3}$.

VI. Conclusions

In this paper, we have investigated cooperative service caching and workload scheduling in mobile edge computing. Based on queuing analysis, we have formulated this problem as a mixed integer nonlinear programming problem, which is proved to have non-polynomial computation complexity. To deal with the challenges of subproblem coupling, computation-communication tradeoff and edge node heterogeneity, we have proposed the two-layer ICE algorithm, with the outer layer updating the service caching policy iteratively based on Gibbs sampling and the inner layer scheduling workloads based on the idea of water filling. Extensive simulations have been conducted to evaluate the effectiveness and convergence of the proposed ICE algorithm, and the impact of edge connectivity has been further analyzed.

Appendix A
Proof of Theorem 1

Let $\boldsymbol{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_1, ..., \boldsymbol{a}_M\}$ be the caching decision space of edge nodes, and in each iteration, a random edge node n randomly chooses a caching decision from $\boldsymbol{A}$. With Algorithm 1 iterating over the edge nodes and the caching decision space, the *edge caching policy* $\boldsymbol{C}$ evolves as a N-dimension Markov chain, in which each dimension represents the caching decision of each edge node. For the convenience of presentation, we analyze the scenario with 2 edge nodes, and the 2-dimension Markov chain is denoted as $\langle \boldsymbol{c_1}, \boldsymbol{c_2} \rangle$. In each iteration, one randomly selected edge node n ($n \in \mathbb{N}$) virtually changes its current caching decision to a random caching decision from $\boldsymbol{c_n}$, thus there is

$$\begin{aligned} \Pr(\langle c_1^*, c_2 \rangle \mid \langle c_1, c_2 \rangle) &= \frac{e^{-\frac{y(\langle c_1^*, c_2 \rangle)}{\omega}}}{NM(e^{-\frac{y(\langle c_1^*, c_2 \rangle)}{\omega}} + e^{-\frac{y(\langle c_1, c_2 \rangle)}{\omega}})} \\ \Pr(\langle c_1, c_2^* \rangle \mid \langle c_1, c_2 \rangle) &= \frac{e^{-\frac{y(\langle c_1, c_2^* \rangle)}{\omega}}}{NM(e^{-\frac{y(\langle c_1, c_2^* \rangle)}{\omega}} + e^{-\frac{y(\langle c_1, c_2 \rangle)}{\omega}})}, \end{aligned} \tag{18}$$

where $y(\langle c_1, c_2 \rangle)$ is the objective value when the caching policy is $\langle c_1, c_2 \rangle$. In this scenario, $N = 2$. Denote by $\pi(\langle c_1, c_2 \rangle)$ the stationary probability distribution of caching policy $\langle c_1, c_2 \rangle$, then $\pi(\langle c_1, c_2 \rangle)$ can be derived by the fine stationary condition of the Markov chain as

$$\begin{aligned} &\pi(\langle a_1, a_1 \rangle) \Pr(\langle a_1, a_m \rangle \mid \langle a_1, a_1 \rangle) \\ &= \pi(\langle a_1, a_m \rangle) \Pr(\langle a_1, a_1 \rangle \mid \langle a_1, a_m \rangle). \end{aligned} \tag{19}$$

Substitute (18) into (19), it can be derived that

$$\begin{aligned} &\pi(\langle a_1, a_1 \rangle) \times \frac{e^{-\frac{y(\langle a_1, a_m \rangle)}{\omega}}}{NM(e^{-\frac{y(\langle a_1, a_m \rangle)}{\omega}} + e^{-\frac{y(\langle a_1, a_1 \rangle)}{\omega}})} \\ &= \pi(\langle a_1, a_m \rangle) \times \frac{e^{-\frac{y(\langle a_1, a_1 \rangle)}{\omega}}}{NM(e^{-\frac{y(\langle a_1, a_m \rangle)}{\omega}} + e^{-\frac{y(\langle a_1, a_1 \rangle)}{\omega}})}. \end{aligned} \tag{20}$$

It can be observed that Eq. (20) is symmetric and can be balanced if $\pi(\langle c_1, c_2 \rangle)$ has the form of $\pi(\langle c_1, c_1 \rangle) = \gamma e^{-\frac{y(\langle c_1, c_2 \rangle)}{\omega}}$, where γ is a constant. Let Φ be the caching policy space. To ensure $\sum_{\langle c_1, c_2 \rangle \in \Phi} \pi(\langle c_1, c_2 \rangle) = 1$, the stationary probability distribution $\pi(\langle c_1, c_2 \rangle)$ should be given as

$$\pi(\langle c_1, c_2 \rangle) = \frac{e^{-\frac{y(\langle c_1, c_2 \rangle)}{\omega}}}{\sum_{\left\langle c_1^f, c_2^f \right\rangle \in \Phi} e^{-\frac{y(\left\langle c_1^f, c_2^f \right\rangle)}{\omega}}} \tag{21}$$

Eq. (21) can be rewritten as

$$\pi(\langle c_1, c_2 \rangle) = \frac{1}{\sum_{\left\langle c_1^f, c_2^f \right\rangle \in \Phi} e^{\frac{y(\langle c_1, c_2 \rangle) - y(\left\langle c_1^f, c_2^f \right\rangle)}{\omega}}}. \tag{22}$$

Let $\langle c_1^*, c_2^* \rangle$ be the globally optimal solution that minimizes the objective value, i.e., $y(\langle c_1^*, c_2^* \rangle) \leq y(\left\langle c_1^f, c_2^f \right\rangle)$ for any $\left\langle c_1^f, c_2^f \right\rangle \in \Phi$. It can be concluded that $\pi(\langle c_1^*, c_2^* \rangle)$ increases as ω decreases, and $\pi(\langle c_1^*, c_2^* \rangle) \to 1$ when $\omega \to 0$.

Appendix B
Proof of Theorem 2

An optimization problem should satisfy that the objective function and the inequation constraint functions are convex, and the equation constraint function is affine over the decision variables. It is easy to identify that the inequation and equation constraint functions satisfy these conditions. We just need to prove the convexity of the objective function.

In Eq. (12), it is intuitive that $\sum_{s \in S} (\lambda_{os} d_{\text{cloud}} + w_s \lambda_{os})$ and $\sum_{s \in S} \sum_{n \in N} \frac{\max\{\lambda_{ns} A_s - A_{ns}, 0\}}{A_s} d_n$ are convex over $\boldsymbol{\Lambda}$. Let $x(\boldsymbol{\Lambda}) = \sum_{s \in S} \sum_{n \in N} \frac{\lambda_{ns}}{\mu_{ns} - \lambda_{ns} A_s}$. Denote by $H = [h_{mn}]_{m \times n}$ the Hessian matrix of $x(\boldsymbol{\Lambda})$, and $h_{mn} = \frac{2\mu_{ns} A_s}{(\mu_{ns} - A_s \lambda_{ns})^3}$ when $m = n$ ($\forall m \in \mathbb{N}$, $n \in \mathbb{N}$), otherwise, $h_{mn} = 0$. It can be noticed that H is a positive definite matrix, and $x(\boldsymbol{\Lambda})$ is convex over $\boldsymbol{\Lambda}$ [28]. The objective function $f(\boldsymbol{\Lambda})$ is the sum of several convex functions over $\boldsymbol{\Lambda}$, so $f(\boldsymbol{\Lambda})$ is also convex over $\boldsymbol{\Lambda}$. Thus, we can conclude that problem **P2** is a convex optimization problem over the workload scheduling policy $\boldsymbol{\Lambda}$.

Acknowledgment

This work was supported by the National Science Foundation of China (61902036, 61922017, 61801011) and China Postdoctoral Science Foundation 2019M650589.

REFERENCES

[1] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, 2010, pp. 49–62.

[2] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. ACM European Conference on Computer Systems (EuroSys'11)*, 2011, pp. 301–314.

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

[4] Cisco, "Cisco global cloud index: Forecast and methodology, 2016-2021," *White Paper*, 2018.

[5] ETSI. Mobile edge computing (mec); framework and reference architecture, etsi gs mec 003 v1.1.1, 2016.

[6] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, "Just-in-time provisioning for cyber foraging," in *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'13)*, 2013, pp. 153–166.

[7] 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 (INFOCOM'18)*, 2018, pp. 207–215.

[8] 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.

[9] 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 *IEEE International Conference on Distributed Computing Systems (ICDCS'18)*, 2018, pp. 365–375.

[10] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE Conference on Computer Communications (INFOCOM'19)*, 2019, pp. 1279–1287.

[11] B. Liang, *Mobile edge computing*, V. W. S. Wong, R. Schober, D. W. K. Ng, and L.-C. Wang, Eds. Cambridge University Press, 2017.

[12] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. S. Shen, "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *IEEE Transactions on Cloud Computing*, 2019.

[13] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *IEEE International Conference on Distributed Computing Systems (ICDCS'17)*, 2017, pp. 1281–1290.

[14] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *IEEE Conference on Computer Communications (INFOCOM'17)*, 2017, pp. 1–9.

[15] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2016, pp. 1–9.

[16] Y. Cui, J. Song, K. Ren, M. Li, Z. Li, Q. Ren, and Y. Zhang, "Software defined cooperative offloading for mobile cloudlets," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1746–1760, 2017.

[17] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *IEEE Conference on Computer Communications (INFOCOM'10)*, 2010, pp. 1–9.

[18] 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.

[19] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 915–929, 2019.

[20] G. Dán and N. Carlsson, "Dynamic content allocation for cloud-assisted service of periodic workloads," in *IEEE Conference on Computer Communications (INFOCOM'14)*, 2014, pp. 853–861.

[21] I. Hou, T. Zhao, S. Wang, K. Chan *et al.*, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'16)*, 2016, pp. 291–300.

[22] 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 Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2016.

[23] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 762–772, 2013.

[24] 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*, vol. 25, no. 3, pp. 1635–1648, 2017.

[25] 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 (INFOCOM'19)*, 2019, pp. 10–18.

[26] S. M. Lynch, *Introduction to Applied Bayesian Statistics and Estimation for Social Scientists*. Springer, 2007.

[27] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

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