Huang *et al. Journal of Cloud Computing* (2023) 12:132
https://doi.org/10.1186/s13677-023-00506-7

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH **Open Access**

Stateless Q-learning algorithm for service caching in resource constrained edge environment

Binbin Huang[1], Ziqi Ran[1], Dongjin Yu[1*], Yuanyuan Xiang[1], Xiaoying Shi[1], Zhongjin Li[1] and Zhengqian Xu[1]

Abstract

In resource constrained edge environment, multiple service providers can compete to rent the limited resources to cache their service instances on edge servers close to end users, thereby significantly reducing the service delay and improving quality of service (QoS). However, service providers renting the resources of different edge servers to deploy their service instances can incur different resource usage costs and service delay. To make full use of the limited resources of the edge servers to further reduce resource usage costs, multiple service providers on an edge server can form a coalition and share the limited resource of an edge server. In this paper, we investigate the service caching problem of multiple service providers in resource constrained edge environment, and propose an independent learners-based services caching scheme (ILSCS) which adopts a stateless Q-learning to learn an optimal service caching scheme. To verify the effectiveness of ILSCS scheme, we implement COALITION, RANDOM, MDU, and MCS four baseline algorithms, and compare the total collaboration cost and service latency of ILSCS scheme with these of these four baseline algorithms under different experimental parameter settings. The extensive experimental results show that the ILSCS scheme can achieve lower total collaboration cost and service latency.

Keywords Edge environment, service caching, Stateless Q-learning, Collaboration cost, Service latency

Introduction

With the explosive growth of smart end devices, various latency-sensitive network services provided by different service providers [1], such as virtual reality (VR), real-time navigation, and interactive online games [2], have emerged, which bring great convenience to people's lives. Traditionally, the service instances corresponding to these latency-sensitive services are deployed on the remote cloud datacenters. When a large number of end users frequently access these service instances, it will pose a long service latency and a huge traffic burden on the core networks [3, 4]. To address this problem, edge computing as a new computing paradigm, which sinks the computation, bandwidth and storage resources from remote cloud to the edge servers close to end users, provide a promising solution. In edge computing environment, service providers can rent Virtual Machines (VMs) encapsulating the computation and bandwidth resources of edge servers to deploy their service instances, thereby greatly reducing the service latency and improve the quality of service. However, the resources of edge servers are limited, and service providers renting the limited resource of different edge servers to cache service instances can incur different resource usage costs and service latency [5]. To reduce service latency and make full use of the limited resources to further reduce resource usage costs, multiple service providers can share leased VMs with other service providers.

There are some existing studies on service caching problem in edge environment [6–9]. In particular, Xia

*Correspondence:
Dongjin Yu
yudj@hdu.edu.cn
[1] School of Computer, Hangzhou Dianzi University, Hangzhou 310018, China

et al. [6] formulate the edge data caching problem into a constrained optimization problem and then adopt an integer programming and an approximation algorithm to solve this problem. Its main objective is to minimize the data caching cost and maximize the reduction in service latency. However, this work only considers collaborative service caching between adjacent edge servers in static scenarios. To address these problems, Xia et al. [7] propose a Lyapunov optimization based online algorithm to solve the dynamic collaborative edge data caching problem, aiming at minimizing the overall system cost. However, this work mainly focuses on service cost optimization without considering service latency reduction. In order optimize the long-term utility defined as the weighted sum of the service cost and the service latency reduction, Huang et al. [8] propose a utility-aware collaborative service caching scheme to coordinate multiple edge servers to cache service instances. However, all of the above studies mainly focus that in resource constrained edge environment, edge servers cooperate with each other to quickly retrieve the required service instances. They don't consider that multiple service providers can share leased VMs with other service providers to make full use of the limited resource of edge servers and reduce the collaboration cost.

In this paper, we investigate the service caching problem with resource sharing among multiple service providers in resource constrained edge environment. To address this problem, we construct the resource sharing model by multiple service providers, cost model for service provider and service latency model, respectively. Based these models, we further formulate the service caching problem. In order to solve this problem, we propose an independent learners-based service caching scheme (ILSCS) to minimize the collaboration cost and the service latency. The ILSCS scheme adopt a stateless Q-learning algorithm, in which each edge server is treated as an agent, the caching decision of each service instance as a base action, and the inverse of the collaboration costs, which is a function of the service latency and the usage cost of shared resource, as the immediate reward, to learn an optimal service caching policy. In order to verify the effectiveness of the ILSCS scheme, we implement COALITION, RANDOM, MDU and MCS four baseline algorithms. We compare the total collaboration cost and the service latency of ILSCS scheme with these of these four baseline algorithms under different environmental parameters such as service size, number of services, number of edge servers, and storage capacity of edge servers. The related experimental results demonstrate that the ILSCS scheme can achieve lower total collaboration cost and service latency. Our main contributions can be summarized as follows:

(1) We formulate the service caching problem with resource sharing among edge service providers in resource constraint edge environment.
(2) We propose an independent learner-based service caching scheme to minimize the total collaboration cost and the service latency. The ILSCS scheme adopts a stateless Q-learning algorithm to learn an optimal service caching scheme.
(3) We implement four baseline algorithms and conduct extensive experiments to compare the total collaboration cost and the service latency with these of four baseline algorithms. The related experimental results demonstrate that the ILSCS schem can reduce the collaborative cost and the service latency.

We organize the remainder of this paper as follows. We summarize the state-of-the-arts on this topic in Related works. We formulate the service caching problem of multiple service providers in resource constraint edge environment in System model and problem formulation. We describe the proposed ILSCS scheme in The independent learners-based service caching scheme. We conduct extensive experiments and analyse the related experimental result in Experimental evaluation. Finally, we conclude this paper in Conclusions and future work.

Related works

The service caching problem in resource constrained edge environment is a very popular research topic. There are a large number of related studies on this service caching problem [10–20]. According to whether edge servers cooperate with each other, these related studies can be classified two types: service caching without cooperation and service caching with cooperation.

For service caching without cooperation, various approaches including popularity prediction, heuristic approach and etc., are adopted to make service caching decision [10–13]. For example, Du et al. [10] adopted a reduced support vector regression (rSVR) model to predict the popularity of cached files to improve the hit rate of cached files. Compared to the original SVR model, the rSVR model learns only on a smaller reserved subset and requires less storage space. Rim et al. [11] suggested to update caching content based on individual users' short-term content preferences and proposed a content caching strategy based on joint mobility prediction and user prefetching (MPJUP). This strategy reduces the average latency and backhaul load of data fetching by predicting the user's location and the required data. Qi et al. [12] designed a neural network to predict the popularity of content, and based on which a heuristic approach is adopted to optimize active and responsive hybrid caching policies. Its main goal is to improve the overall successful offloading ratio of the mobile edge network. Wang et al. [13] modeled the caching problem as a Markov decision process and propsosed a distributed cache replacement strategy base on Q-learning to minimize the transmission cost. However, this paper mainly considers to optimize the traffic and does not focus the service cost and the service latency. In additon, this paper does not consider multiple service providers to share the limited resources of edge servers.

For service caching with cooperation, some related studies design various cooperation schemes to coordinate multiple edge servers to cache service instances [14–20]. For example, Ahani et al. [14] proposed an optimal content caching scheme in a time-slot system with delivery deadline and cache capacity constraints, the objective of which is to minimize the cost of the backhaul link load. Kim et al. [15] proposed a distributed edge caching scheme to reduce the content delivery delay in edge network with limited storage, content popularity, content placement and access capacity. Gu et al. [16] formulate a cooperative edge caching problem to be a non-cooperative game model and proposed a cooperative edge caching framework, aiming to reduce data transfer latency, relieve data traffic on the backbone network and reduce the workload of cloud servers. Kim et al. [17] proposed a cooperative edge caching approach based on deep reinforcement learning to promote cooperation among edge servers and improve the hit ratio of the system. Ren et al. [18] proposed a cooperative caching scheme based on game theory to make caching decision. Its main goal is to minimize the average latency of acquiring content. However, all of the above studies mainly consider the service caching problem with service providers exclusive resources. They don't consider multiple service providers to share leased VMs with other service providers. Its main goal of which is to minimize the resource usage cost of all service providers. Song et al. [19] proposed a distributed algorithm based on alternating direction method of multipliers to jointly optimize the content caching in cooperative base stations, aiming at reducing cost of content retrieving. This paper does not focus the service latency. Lu et al. [20] formulated the service placement problem as a mixed-integer linear programming problem. To address this problem, this paper proposed a deep reinforcement learning (DSP-DRL) based decentralized dynamic placement framework to minimize the latency. However, this paper does not consider the cost. In addition, all of these above studies do not consider multiple service providers to share the limited resources of edge servers.

System model and problem formulation

In this section, we first introduce the system model. Then we present the resource sharing model by multiple service providers, cost model for service provider, and utility model for service caching in resource constrained edge environment, respectively. Finally, we formulate the service caching problem of multiple service providers in resource constrained edge environment. Each service provider is allowed to share its VM with others when the VM is idle. The key notation used throughout this paper are listed in Table 1.

Table 1 Key notation

Symbols	Semantics
eNB_i	edge server eNB_i
C_i	the computational capacity of edge server eNB_i
B_i	The bandwidth capacity of edge server eNB_i
S_i	The storage capacity of the edge server eNB_i
$VM_{i,j}$	The jth VM in edge server eNB_i
$C_{i,j}$	The computational capacity of the VM $VM_{i,j}$
$B_{i,j}$	The bandwidth capacity of the VM $VM_{i,j}$
SE_k	The service instance of service provider SP_K
W_k	the workload of the computation request processed by corresponding service instances SE_k
D_k	the size of the input data required by the computation request
g_i	The coalition on the edge server eNB_i
$c_{i,j}$	the cost of the service provider occupying the VM $VM_{i,j}$ alone
d_k^{CL}	The service latency of the computation request processed by the service instance SE_k in the central cloud CL
$d_{i,j,k}$	The service latency of the computation request processed by the service instance SE_k cached on VM $VM_{i,j}$ of edge server eNB_i
$d_{i,j,k}^{exe}$	The service execution latency of the computation requests processed by service instance SE_k cached on VM $VM_{i,j}$ of edge server eNB_i
$d_{i,k}^{tran}$	The transfer time of input data required by service instance SE_k
v_k	Delay preference weight

System model

As shown in Fig. 1, we mainly consider an edge environment consisting of n edge servers $eNB = \{eNB_1, \ldots, eNB_i, \ldots, eNB_n\}$ and a central cloud CL. These edge servers are deployed near the end users. Each edge server eNB_i can be represented by a three-tuple $eNB_i = < C_i, B_i, S_i >$, in which C_i, B_i and S_i denote the computational capacity, bandwidth capacity, and storage capacity of the edge server eNB_i, respectively. These resources of edge server eNB_i can be encapsulated to be m VMs. The set of m VMs can be denoted by $VM_i = \{VM_{i,1}, \ldots, VM_{i,j}, \ldots, VM_{i,m}\}$, in which $VM_{i,j}$ denotes the j th VM in edge server eNB_i. Each VM $VM_{i,j}$ can be denoted by a two-tuple $VM_{i,j} = < C_{i,j}, B_{i,j} >$, in which $C_{i,j}$ denotes the computational capacity of the VM $VM_{i,j}$, and $B_{i,j}$ denotes the bandwidth capacity of the VM $VM_{i,j}$. The central cloud CL hosts a set of original service instances that are to be cached to the VMs of edge servers. Due to the limited resources of edge servers, multiple service providers may compete to rent the limited resources to deploy their service instances [21].

Resource sharing model by multiple service providers

In our edge environment, there are K service providers $SP = \{SP_1, \ldots, SP_k, \ldots, SP_K\}$ and each service provider has a service instance. The set of these service instances can be denoted by $SE = \{SE_1, \ldots, SE_k, \ldots, SE_K\}$. The service instance SE_k of the k th service provider SP_k can be denoted by a two-tuple $SE_k = < W_k, D_k >$, in which W_k denotes the workload of the computation request processed by corresponding service instances SE_k, D_k denotes the size of the input data required by the computation request. Each service instance SE_k have a set of user requests to process. If the k th service provider SP_k caches its service instance SE_k on VM $VM_{i,j}$ of edge server eNB_i, the user requests will be redirected to the edge server eNB_i to process. Otherwise, the user requests will be fulfilled by the original service instance in the central cloud CL. Each service provider provides services with relatively stable performance and has a stable users base. The users of a service provider will not move to other service providers in the short term. To improve the quality of service (QoS) and keep the user base, service providers cache their service instances to edge servers nearby end users [22]. However, caching service instances on the edge servers greatly increases the service cost of service providers. To reduce service cost, different service providers can cache their service instances to different VMs on the same edge server for resource sharing. Moreover, when the VM occupied by service provider is idle, it can also be shared with other

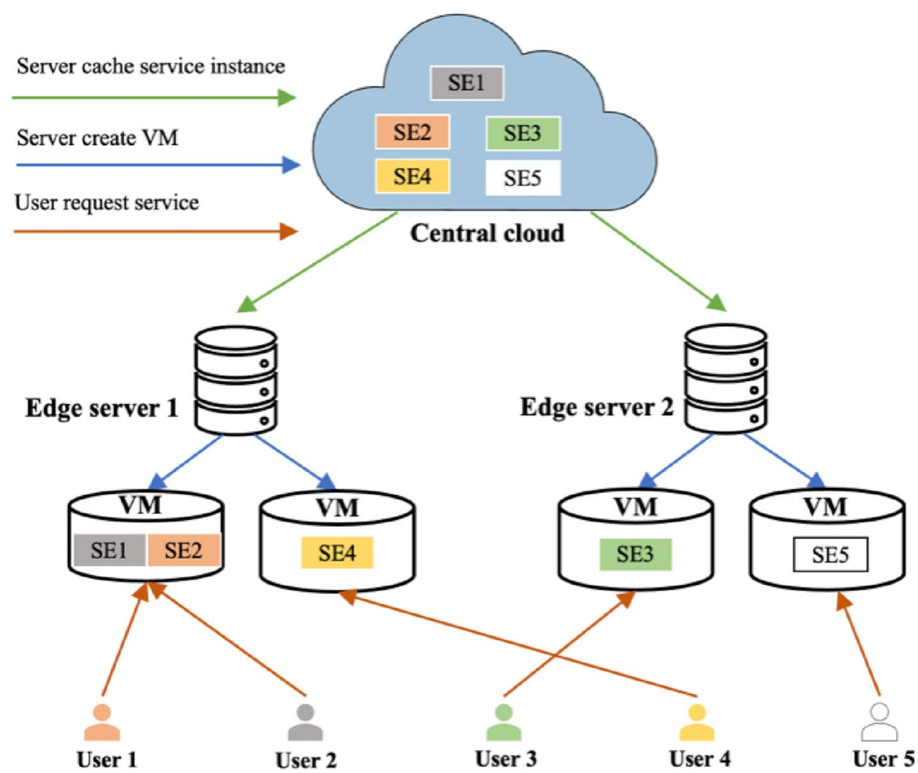


Fig. 1 An example of service caching in edge environment

service providers, and thereby greatly reducing the service cost of service providers [23].

All service providers on the same edge server are referred as a coalition. The coalition on the edge server eNB_i can be denote by g_i. Since the storage resource of each edge server is limited, the sum of the size of input data required by the computation requests corresponding to the service instances in the coalition cannot exceed the storage capacity of the edge server, i.e., $\sum_{SP_{\mathrm{k}} \in g_i} D_{\mathrm{k}} \leq S_i$. Each service provider can apply to join in a coalition. Each coalition has an agent which decides whether service provider's applying is accepted or not. When it is accepted, the agent further assigns the service instance to an optimal VM according to the resources of different VMs.

Cost model for service provider

To cache service providers' service instances on edge servers incurs additional service costs [24]. When a VM only cache a service instance, the cost to rent this VM is undertaken by the corresponding service provider. When a VM is shared by multiple service instances, the cost will be shared among the cached instances [25]. We model the cost of a service provider occupying a VM alone and the cost of a service provider in a coalition sharing a VM, respectively.

When a service instance SE_k is cached on a VM $VM_{i,j}$, the computation and bandwidth resources of the VM $VM_{i,j}$ are exclusive to the service instance SE_k. The cost of using per unit of computation resource of the edge server eNB_i can be denoted by c_i^p. The cost of using per unit of bandwidth resource of the edge server eNB_i can be denoted by b_i^p. Therefore, the exclusive resource usage cost incurred by the service provider SP_k for exclusive ownership of the VM $VM_{i,j}$ can be calculated by Eq. (1):

$$c_{i,j} = c_i^p \cdot C_{i,j} + b_i^p \cdot B_{i,j} \tag{1}$$

It is note that the usage cost of exclusive resource is referred as its default cost.

When multiple service providers on an edge server form a coalition, the computation and bandwidth resources of the edge server can be shared by these service providers. Since each service provider is self-interested, we adopt the cost policy proposed in the literature [26] to ensure the stability of the formed coalition. When service provider SP_k joins in the coalition g_i and caches its service instance SE_k on $VM_{i,j}$ in coalition g_i, the service provider SP_k shares the computation and bandwidth resources of edge server eNB_i with other service providers in coalition g_i and the usage cost $p_{k,j}(g_i)$ of shared resource can be calculated by Eq. (2):

$$p_{k,j}(g_i) = \frac{c_{i,j}}{\sum_{SP_{k'} \in g_i} c_{i,j'}} c(g_i) \tag{2}$$

where $\sum_{SP_{k'} \in g_i} c_{i,j'}$ denotes the sum of the default cost of the service providers in coalition g_i. $c(g_i) = c_i^p \cdot C_i + b_i^p \cdot B_i$ denotes the cost of the edge server eNB_i.

Service latency model

The service latency is defined to be the sum of service execution time and data transfer time. The service latency of the computation request processed by the service instance cached in edge server is very different from that by service instance cached the service in the central cloud CL. [27]. The service latency of the computation request processed by the service instance SE_{k} in the central cloud CL can be denoted by d_k^{CL}. The service latency of the computation request processed by the service instance SE_{k} cached on VM $VM_{i,j}$ of edge server eNB_i can be denoted by $d_{i,j,\mathrm{k}}$. The service latency is composed of service execution delay and data transfer time. The service execution latency of the computation requests processed by service instance SE_{k} cached on VM $VM_{i,j}$ of edge server eNB_i can be calculated by $d_{i,j,k}^{exe} = W_{\mathrm{k}}/C_{i,j}$, where $C_{i,j}$ denotes the computing capacity of $VM_{i,j}$. The transfer time of input data required by service instance SE_{k} can be denoted by $d_{i,k}^{tran}$. Therefore, the service latency $d_{i,j,\mathrm{k}}$ can be denoted by calculated by Eq. (3).

$$d_{i,j,\mathrm{k}} = d_{i,k}^{tran} + d_{i,j,k}^{exe} \tag{3}$$

Since the edge server is closer to the end user than the central cloud, the service latency of the computation request processed by the service instance cached in the edge servers is usually much smaller than that by service instance cached in the central cloud, expressed as $d_{i,j,\mathrm{k}} \ll d_k^{CL}$.

Utility model

To minimize the service cost and the service latency, the utility function can be defined as the weighted sum of the service cost and service latency. The utility obtained by the service instance SE_{k} occupying the VM $VM_{i,j}$ of edge server eNB_i alone is defined as the default utility, which can be denoted by $u_{i,j,k}^{dft}$. The utility obtained by the service instance SE_{k} sharing the resources of edge server eNB_i with other service providers in coalition g_i is defined as the collaboration utility, which can be denoted by $u_{i,j,k}^{coll}$. Different service instances have different delay sensitivities. The importance of the service instance SE_{k} can be adjusted by the weighted v_{k}. Therefore, the default utility and the collaboration utility can be calculated by Eq. (4) and Eq. (5), respectively.

$$u_{i,j,k}^{dft} = v_{\mathrm{k}} \cdot \left(d_k^{CL} - d_{i,j,\mathrm{k}}\right) - c_{i,j} \quad (4)$$

$$u_{i,j,k}^{coll} = v_{\mathrm{k}} \cdot \left(d_k^{CL} - d_{i,j,\mathrm{k}}\right) - p_{k,j}(g_i) \quad (5)$$

A service provider can cache its service instance to a VM $VM_{i,j}$ to maximize its default utility. To further reduce the service cost and obtain greater utility, the service provider can share the resources of VM $VM_{i,j}$ with other service providers in coalition g_i. The additional utility obtained by service provider SP_{k} joining in the coalition g_i can be denoted by $u_{i,j,k}$, which can be calculated by Eq. (6):

$$u_{i,j,k} = u_{i,j,k}^{coll} - u_{i',j',k}^{dft} = c_{i',j'} - (p_{k,j}(g_i) - v_{\mathrm{k}} \cdot d_{i',j',\mathrm{k}} + v_{\mathrm{k}} \cdot d_{i,j,\mathrm{k}}) \quad (6)$$

where $u_{i',j',k}^{dft}$ is the maximum default utility that can be obtained by the service provider SP_{k}. Each service provider with occupying the VM $VM_{i',j'}$ of edge server $eNB_{i'}$ alone has a default service cost $c_{i',j'}$. $p_{k,j}(g_i) - v_{\mathrm{k}} \cdot d_{i',j',\mathrm{k}} + v_{\mathrm{k}} \cdot d_{i,j,\mathrm{k}}$ can reflect the collaboration cost, which can be denoted by $c_{i,j,k}^{coll}$.

Problem formulation

Different edge servers have different computing resources and bandwidth resources [28]. Therefore, the service latency of the computation request processed by different edge servers vary greatly. Moreover, a service provider choosing different edge servers to form a coalition also greatly affects its service cost. Therefore, with limited computing and bandwidth resources of edge servers, the main goal of service caching is to minimize the sum of the collaboration costs of all service providers [29]. Here, we formulate the service caching problem as follows:

$$\text{Minimize}: \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{K} a_{i,j,k} \cdot c_{i,j,k}^{coll} \quad (7)$$

$$\text{Subject to}: \sum_{SP_{\mathrm{k}} \in g_i} D_{\mathrm{k}} \leq S_i \quad (8)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{K} a_{i,j,k} = 1 \quad (9)$$

$$a_{i,j,k} \in \{0, 1\} \quad (10)$$

where $a_{i,j,k}$ denotes whether the service instance SE_{k} of service provider SP_{k} is cached on VM $VM_{i,j}$ of edge server eNB_i. If $a_{i,j,k} = 1$, it denotes that the service instance SE_k is cached in $VM_{i,j}$ of the edge server eNB_i. Otherwise, it means that the service instance SE_k is not cached in $VM_{i,j}$ of the edge server eNB_i. Equation (8) ensures that the sum of the size of the input data required by the computation requests processed by corresponding service instances in the edge server does not exceed the maximum storage capacity S_i of the edge server eNB_i. Equation (9) denotes that the service instance SE_{k} of service provider SP_{k} is only cached on VM $VM_{i,j}$ of edge server eNB_i. Service provider SP_{k} no longer need to cache its service instance SE_{k} on other edge servers, which increases the resource cost.

The independent learners-based service caching scheme

To solve the service caching problem in a resource constrained edge environment, we adopt a stateless Q-learning algorithm, and design an independent learners-based service caching scheme. In this section, we first introduce the stateless Q-learning algorithm. Then, we define the action space and reward function of the service caching problem. Finally, we describe the independent learners-based service caching scheme in detail.

Stateless Q-learning algorithm

Q-learning algorithm is a simple and easy to understand reinforcement learning algorithm [30]. The Q value is the expected reward obtained after taking a specific action at a specific state. The Q value can be used to measure the effectiveness of the actions. The Q-learning algorithm can learn an estimated Q-value obtained by taking each action at each state. However, the environment state is sometimes only releated to actions and not to states [31]. Therefore, the Q-learning algorithm can be further simplified to a stateless Q-learning algorithm [32]. In our problem model, the environment state is manly related to the caching action of service instances. Therefore, a stateless Q-learning algorithm is adopted to solve the service caching problem. Each edge server eNB_i is treated as an agent i, its caching decision as an action a_i, the inverse of the sum of the collaboration costs of all service instances cached in edge server eNB_i as the immediate reward R_i. The expected reward obtained by each agent i performing action a_i in next time period can be denoted $Q_i(a_i)$. For the stateless Q-learning algorithm, the expected reward $Q_i(a_i)$ can be updated by Eq. (11) [32]:

$$Q_i(a_i) \leftarrow Q_i(a_i) + \lambda_T \cdot (R_i(\tau) - Q_i(a_i)) \quad (11)$$

where λ_T denotes the learning rate.

Action space and reward function

The caching action $a(\tau)$ at the time step τ can be defined by the Eq. (12):

$$a(\tau) = (a_1(\tau), \ldots, a_i(\tau), \ldots, a_n(\tau)) \tag{12}$$

where $a_i(\tau) = (a_{i,1,1}(\tau), \ldots, a_{i,j,k}(\tau), a_{i,m,K}(\tau))$ is a vector that indicates whether the K service instances are cached on the edge server eNB_i.$a_{i,j,k}(\tau)$ indicates whether service instance SE_{k} is cached on $VM_{i,j}$ at the time step τ. If $a_{i,j,k}(\tau) = 1$, the service instance SE_{k} is cached on $VM_{i,j}$ of edge server eNB_i at the time step τ. Otherwise, $a_{i,j,k}(\tau) = 0$ indicates that service instance SE_{k} is not cached on $VM_{i,j}$ of the edge server eNB_i at the time step τ.

The actions $a(\tau)$, $a_i(\tau)$, and $a_{i,j,k}(\tau)$ are called super action, joint action and base action, respectively. Their action spaces' sizes are $n \cdot 2^{MK}$, 2^{MK} and 2, respectively. Sine the action spaces' sizes of the super action and joint action are exponential, the Q-learning algorithm needs an exponential number of iterations to go through all actions and learn their Q values, which is clearly infeasible. To address this problem, refer to the literature [32], the Q-value $Q_{i,j,k}\left(a_{i,j,k}\right)$ of each base action $a_{i,j,k}$ is first learned. Then, according to the Q-values of all base action, the Q-value of the super action can be obtained. Therefore, the action space of super action can be greatly reduced to that of base action. After the agent i of edge server eNB_i performing the base action $a_{i,j,k}(\tau)$, the expected reward $Q_{i,j,k}\left(a_{i,j,k}\right)$ can be obtained. Based on the Q-value $Q_{i,j,k}\left(a_{i,j,k}\right)$ of basic action $a_{i,j,k}$, the Q-value $Q_i(a_i)$ of the super action a_i can be further calculated. The expected reward $Q_{i,j,k}\left(a_{i,j,k}\right)$ of base action $a_{i,j,k}(\tau)$ can be updated by Eq. (13):

$$Q_{i,j,k}\left(a_{i,j,k}\right) \leftarrow Q_{i,j,k}\left(a_{i,j,k}\right) + \frac{1}{C_{i,j,k}\left(a_{i,j,k}\right) + 1} \cdot \left(R_{i,j,k}(\tau) - Q_{i,j,k}\left(a_{i,j,k}\right)\right) \tag{13}$$

where $C_{i,j,k}\left(a_{i,j,k}\right)$ denotes the number of times that service instances SE_{k} is cached on VM $VM_{i,j}$ of edge server eNB_i at time step τ. $R_{i,j,k}(\tau)$ denotes the immediate reward obtained by caching the service instance SE_{k} on VM $VM_{i,j}$ of edge server eNB_i at time step τ. Since the immediate reward $R_{i,j,k}(\tau)$ is the inverse of the cost of the service instance SE_{k} cached on VM $VM_{i,j}$ of edge server eNB_i, it can be denoted by $R_{i,j,k}(\tau) = -a_{i,j,k} \cdot c_{i,j,k}^{coll}$. The estimated reward obtained by caching the service instance SE_{k} on VM $VM_{i,j}$ of edge server eNB_i can be calculated by $Q_{i,j,k} = Q_{i,j,k}(1) - Q_{i,j,k}(0)$. When $a_{i,j,k} = 0$, $R_{i,j,k}(\tau) = 0$. Based on the values of $a_{i,j,k}$ and $R_{i,j,k}(\tau)$, we can further compute the estimated rewards of action $a_{i,j,k} = 0$ and action $a_{i,j,k} = 1$, and obtain $Q_{i,j,k}(0) = 0$ and $Q_{i,j,k} = Q_{i,j,k}(1)$.

Algorithm implementation

To solve this above problem, we propose an independent learners-based services caching scheme (ILSCS). The ILSCS scheme adopt a stateless Q-learning algorithm to learn an optimal service caching with resource sharing among multiple service providers. The detail process of ILSCS scheme can be presented in Algorithm 1. We first initialize all $C_{i,j,k}$ and $Q_{i,j,k}$ to be 0 (line 1). Then we calculate the immediate reward $R_{i,j,k}(\tau)$ obtained by taking base action $a_{i,j,k}(\tau)$ (line 4). According to the immediate reward $R_{i,j,k}(\tau)$, we further update the corresponding $Q_{i,j,k}$ and $C_{i,j,k}$ (line 5–6).Base on all base actions $a_{i,j,k}(\tau)$, we can further to find an optimal super action $a^*(\tau)$. Referring to the literature [32, 33], the problem to find the optimal super action a^* can be converted to be a 0–1 backpack problem. It can be formulated as follows:

$$\text{Maximize} : \sum\nolimits_{i=1}^{n} \sum\nolimits_{j=1}^{m} \sum\nolimits_{k=1}^{K} a_{i,j,k} \cdot Q_{i,j,k} \tag{14}$$

Subject to:(8)(9)(10).

The 0–1 knapsack problem is a classical NP-hard problem [34]. It is very difficult to find the optimal super action a^*. In this paper, we adopt a greedy algorithm to solve the 0–1 knapsack problem and find an approximate optimal solution. We first calculate $Q_{i,j,k}/D_{\mathrm{k}}$, where $i = 1, 2, \ldots, n$,$j = 1, 2, \ldots, m$,$k = 1, 2, \ldots, k$. Then, we sort $Q_{i,j,k}/D_{\mathrm{k}}$ in non-increasing order (line 9). According to the order of $Q_{i,j,k}/D_{\mathrm{k}}$, we sequentially perform the corresponding service caching actions. Specifically, the service caching action $a_{i,j,k}(\tau)$ corresponding to $Q_{i,j,k}/D_{\mathrm{k}}$ is to cache the service instance SE_{k} in the $VM_{i,j}$ of edge server eNB_i. For the service caching action corresponding to the last 10% of $Q_{i,j,k}/D_{\mathrm{k}}$, we adopt an epsilon-greedy algorithm to select a service caching action (line 14). It means that we choose the service caching action corresponding to the last 10% of $Q_{i,j,k}/D_{\mathrm{k}}$ with probability ε and randomly select a service caching action with probability $1 - \varepsilon$. Otherwise, we perform the service caching action $a_{i,j,k}(\tau)$ corresponding to $Q_{i,j,k}/D_{\mathrm{k}}$ until constraint conditions (8), (9) and (10) are not satisfied (line 15–16). Finally, in order to make service providers join in coalitions to reduce the service cost, the service providers that do not join in any coalition are required to cache their service instances on the edge servers that minimizes their collaboration cost.

Input: Collection of services provided by service providers

Output: Service caching decision

01: $C_{i,j,k}(a_{i,j,k}) = 0, Q_{i,j,k}(a_{i,j,k}) = 0, i = 1,2,\dots,n, j = 1,2,\dots,m, k = 1,2,\dots,\mathrm{K}$;

02: Set $\tau = 0$;

03: **for** $i = 1,2,\dots,n, j = 1,2,\dots,m, k = 1,2,\dots,\mathrm{K}$ **do:**

04: Calculate $R_{i,j,k}(\tau)$;

05: $Q_{i,j,k}(a_{i,j,k}) \leftarrow Q_{i,j,k}(a_{i,j,k}) + \frac{1}{C_{i,j,k}(a_{i,j,k})+1} \cdot (R_{i,j,k}(\tau) - Q_{i,j,k}(a_{i,j,k}))$;

06: $C_{i,j,k}(a_{i,j,k}) \leftarrow C_{i,j,k}(a_{i,j,k}) + 1$;

07: Calculate $Q_{i,j,k}/D_{\mathrm{k}}$;

08: **end for**

09: Sort $Q_{i,j,k}/D_{\mathrm{k}}$ in non-increasing order;

10: Remove service instances on all edge servers;

11: **while** there are service instances that have not been cached yet **do:**

12: **if** the service instance SE_{k} is not yet cached **then:**

13: **if** $Q_{i,j,k}/D_{\mathrm{k}}$ is in the last 10% **then:**

14: Perform the action corresponding to the last 10% of $Q_{i,j,k}/D_{\mathrm{k}}$ with probability ε, and randomly select a service caching action with probability $1-\varepsilon$;

15: **if** $\sum_{SP_{\mathrm{k}'} \in g_i} D_{\mathrm{k}'} + D_{\mathrm{k}} \le S_i$ **then:**

16: Cache service instance SE_{k} to $VM_{i,j}$;

17: **end while**

18: The service providers that do not join in any coalition are required to cache their service instances on the edge servers that minimizes their collaboration cost;

19: Set $\tau \leftarrow \tau + 1$;

20: Back to Step 03

Algorithm 1. Independent learners-based service caching scheme (ILSCS)

Experimental evaluation

In order to evaluate the effectiveness of our proposed ILSCS scheme, we conduct extensive experiments to compare ILSCS scheme against COALITION, RANDOM, MDU, and MCS four baseline algorithms under different experimental settings. In this section, we first present the experiment parameters setting. Then we analyze the related experimental results.

Experimental parameter settings

In this paper, the edge environment mainly consists of K service providers, n edge servers and a central cloud CL. Each service provider has a service instance. We set the related experimental parameters referring to literatures [26]. These experimental parameters are described in detail as follows.

(1) The parameter settings for system model: the number n of edge servers is 50 in default. The computation capacity C_i of each edge server eNB_i varies within the range [8000, 16000] MHz. The bandwidth capacity B_i of each edge server eNB_i varies within the range [100, 1000] Mbps. The storage capacity S_i of each edge server eNB_i varies within [200–300] GB. The computation capacity $C_{i,j}$ of VM $VM_{i,j}$ in edge server eNB_i varies within [4000–8000] MHz and its bandwidth capacity $B_{i,j}$ varies within [10–100] Mbps. The usage cost per unit compute resource of each edge server eNB_i is set to [\$0.15, \$0.22]. The usage cost per unit bandwidth resource of each edge server eNB_i varies within [\$0.05, \$0.12]. The transmission delay $d_{i,k}^{tran}$ between the end user and the edge server eNB_i where cache the service instance SE_{k} required by the end user is set to 5-20 ms. The service latency of the computation request processed by the service instance cached in central cloud is set to [50, 100] ms.

(2) The parameter settings for resource sharing model by multiple service providers: the number K of service providers is 80 by default. The size D_{k} of the service instance SE_{k} provided by service provider SP_{k} varies within [30, 50] GB. The workload W_k of the computation request processed by corresponding service instances SE_k are set to [50, 100] MHz. The weighted v_{k} varies within [100, 150].

Experimental analysis

To verify the effectiveness of ILSCS scheme, we implement COALITION, RANDOM, MDU and MCS four baseline algorithms. We compare the performance of ILSCS scheme with that of four baseline algorithms under different experimental parameter settings, and analyze these experimental results.

- ILSCS: This abbreviation standards for independent learners-based service caching scheme. This scheme treats each edge server as an agent, and adopts a stateless Q-learning to learn an optimal service caching policy.
- COALITION [26]: It adopts a distributed and stable game-theoretic mechanism to solve the service caching problem with resource sharing among multiple service providers, aiming at minimizing the social cost of all service providers.
- RANDOM: It randomly selects virtual machines to cache service instances of service providers.
- MDU (max default utility): It caches the service instances of service providers to these virtual machines that maximum their default utility.
- MCS (max coalition size): It caches the service instances of service providers to these edge servers with the most members in the coalition.

Convergence of ILSCS

Figure 2 shows the learning curve of the ILSCS scheme. We can observe from Fig. 2 that the total collaboration cost gradually decreases and tends to be stable with the increase of the learning time (i.e., the number of episodes). It indicates that the ILSCS scheme can learn an optimal service caching strategy that minimize the total collaboration

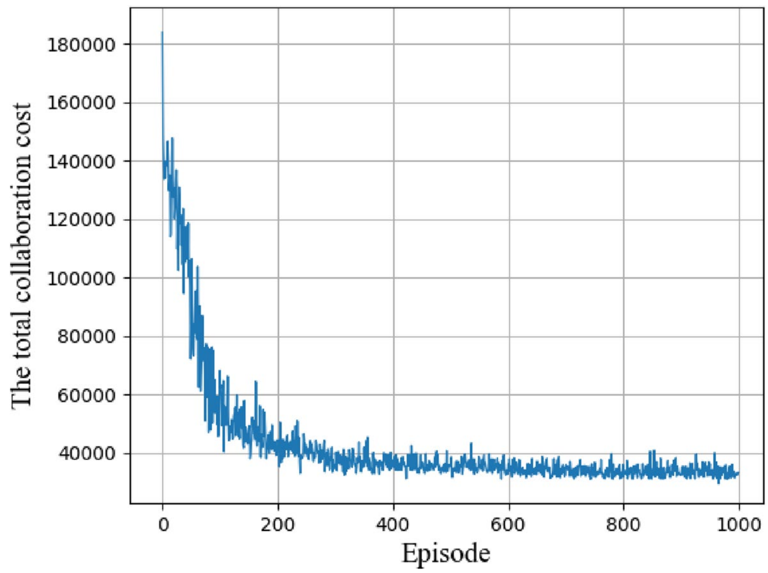
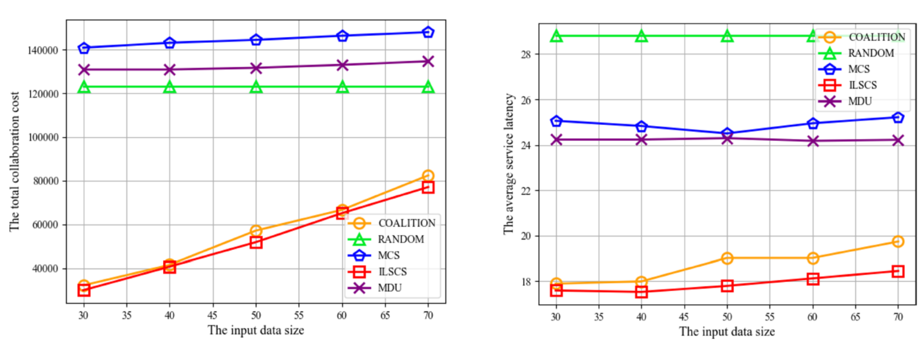


Fig. 2 The learning curve of ILSCS scheme

cost of all service providers. In resource constrained edge environment, each edge server is treated as an agent. Each agent can learn a collaboration caching scheme, that is how multiple service providers on this edge server can share the limited resource of the edge server to cache their service instances, to greatly reduce the resource usage cost.

Impact of the size of the input data required by computation request

To investigate the impact of the size of the input data required by computation request on the total collaboration cost and the average services delay, we vary the input data's size from 30 to 70 GB with the increment of 10 GB. Figure 3 show the impact of different sizes of the input data required by computation request on the total collaboration cost and the average latency. We can observe from the Fig. 3 that the total collaboration cost and the average service latency of ILSCS scheme are lower than these of COALITION, MDU, MCS and RANDOM four algorithms when the size of the service instances gradually increase. That is because that the ILSCS scheme can learn an optimal service caching policy with resource sharing among multiple service provider in a coalition of an edge server, which greatly reducing the collaboration cost and the server latency. Moreover, we can observe from Fig. 3(a) that the total collaboration cost of ILSCS, COALITION, MDU and MCS four algorithms gradually increase with the increase of the input data size. The main reason is that when the input data size increases, the number of service instances cached on an edge server decreases, thereby the number of service providers in coalition of the edge server decreasing. The smaller the number of service providers in coalition of the edge



(a) The total collaboration cost (b) The average services latency

Fig. 3 The impact of the size of input data required by computation request server, the higher the collaboration cost that paid by the members in the coalition, and thereby leading to higher social cost.

The impact of the number of service providers

To examine the impact of the number of service providers on the total collaboration cost and the service delay, we vary the number of service providers from 40, 60, 80, 100 to 120. Figure 4 plot the related experimental result. We can see from Fig. 4(a) that the total collaboration cost of ILSCS, COALITION, MDU, MCS and RANDOM five algorithms gradually increase with the increase of the number of service providers. The main reason for this phenomenon is that the total collaboration cost is the sum of the collaboration cost of all service providers. When the number of service providers increases, the sum of the collaboration cost of all service providers increases as well. Moreover, we can also see from Fig. 4 that the total collaboration cost and the average service latency of ILSCS scheme are lower than these of COALITION, MDU, MCS and RANDOM four baseline algorithms. That is because that the ILSCS scheme can learn an optimal service caching policy once the number of service providers is fixed. On the one hand, the optimal service caching policy can cache service instances on optimal edge servers, which achieve a lower average service latency. On the other hand, the optimal service caching policy enables multiple service provider to share the limited resources of edge servers to cache more service instances, and thereby incurring lower collaboration cost and the average service delay.

The impact of the number of edge servers

Figure 5 illustrates the impact of the number of edge servers on the total collaboration cost and the average service latency. In Fig. 5, we can see that with the number of edge servers varying from 20, 35, 50, 65, to 80, the total collaboration cost and the average service latency of ILSCS scheme gradually decrease. This is due to that with the increase of the number of edge servers, there are more available edge servers that can be selected to cache service instances of service providers. The more available edge servers, the higher probability the service providers have to select more cost-effective edge servers to cache their service instances, and thereby reducing the collaborative cost, alleviating the resource contention and reducing the average service latency. The MCS algorithm selects these edge servers with the most members in the coalition, rather than the edge servers with the highest cost-effective, to cache service instances. Therefore, we can observe that the total collaboration cost and the service latency of MCS algorithm are not relative to the number of edge servers. The RANDOM algorithm randomly selects edge servers to cache their service instances. The more available edge servers, the more scattered the service instances will be cached, and thereby leading to higher collaboration cost. Therefore, the collaboration cost of the RANDOM algorithm gradually increases with the increase of the number of edge servers. In addition, we can observe that the collaboration cost and the service latency of ILSCS scheme are lower than these of COALITION, MDU, MCS and RANDOM four algorithms. The main reason is that the ILSCS scheme can selects edge servers with the highest cost-effective to cache service instances and shares the resources of edge servers among multiple service instances on the same edge server.

The impact of the storage capacities of edge servers

To investigate the impact of the storage capacity of edge server on the total collaboration cost and the service latency, we vary the storage capacities of edge server from 200 GB, 250 GB, 300 GB, 350 GB to 400 GB. Figure 6 plots the related experimental result. We can observe from the Fig. 6(a) that the total

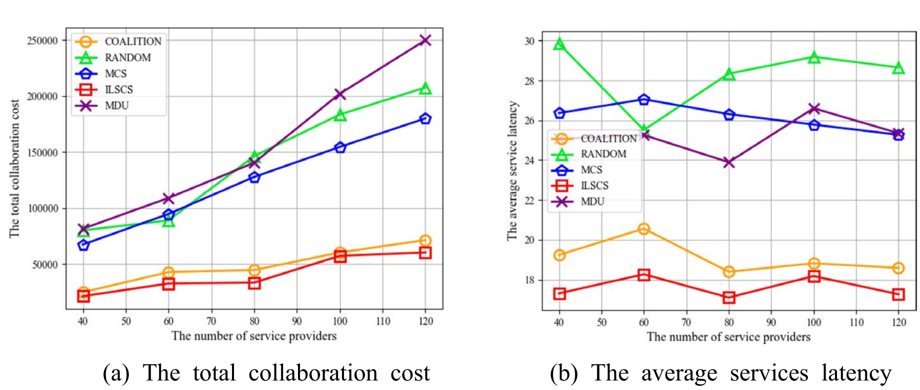
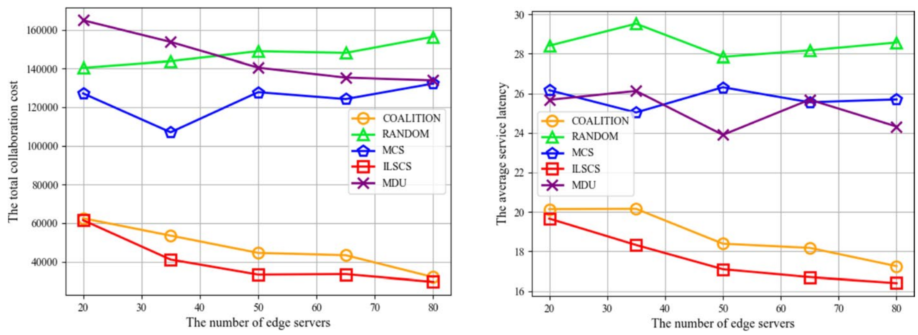


Fig. 4 The impact of the number of service providers



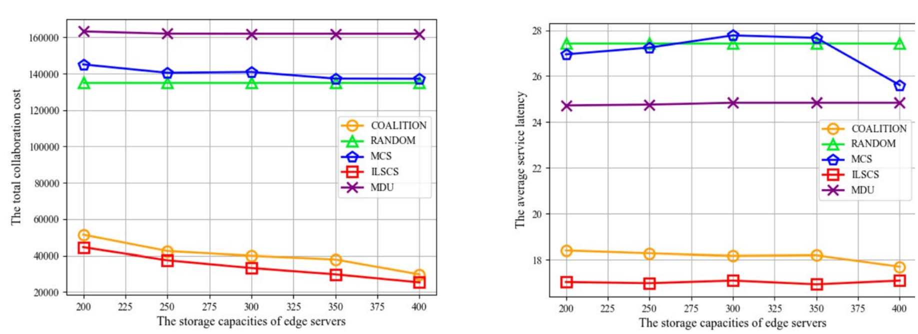
(a) The total collaboration cost (b) The average services latency

Fig. 5 The impact of the number of edge servers

collaboration cost of MCS, COALITION and MRSCS algorithms decreases with the increase of the storage capacities of the edge servers. That is because that the larger the storage capacity of the edge server is, the more service providers can join in a coalition of an edge server, and thereby decreasing the collaboration cost of the service providers in the coalition. The total collaboration costs of RANDOM and MDU algorithms are not affected by the storage capacities of edge servers. In addition, the total collaborative cost of ILSCS and COALITION two algorithms decrease faster than that of RANDOM, MCS and MDU algorithms. This is because when the storage capacity of edge servers increases, the probabilities of resource contention among service providers decreases. With lower resource contention, the service providers can select the high cost-effective edge servers to cache service instances, and thereby greatly decreasing the social cost. Finally, we can further observe from the Fig. 6(a) that the total collaboration cost and the server latency of ILSCS scheme are lower than these of COALITION, MDU, MCS and RANDOM four algorithms. The main reason is discussed in The impact of the number of edge servers.

Conclusions and future work

In this paper, we investigate the service caching problem with resource sharing among multiple service providers in resource constrained edge environment. To address this problem, we first construct system model, resource sharing model by multiple service providers, cost model for service provider, service latency model and utility model, respectively. Then we formulate the service caching problem with resource sharing among multiple service providers. Next, we adopt a stateless Q-learning algorithm to learn an optimal service caching policy. Finally, to validate the effectiveness of our proposed ILSCS scheme, we implement COALITION, RANDOM, MDU and MCS four baseline algorithm, and compare the total collaborative cost and the service latency of our proposed ILSCS scheme to these of four baseline algorithms under different experimental parameter settings such as the size of service instance, the number of service instances, the number of edge



(a) The total collaboration cost (b) The average services latency

Fig. 6 The impact of the storage capacities of edge servers (**a**) The total collaboration cost. (**b**) The average services latency servers, and the storage capacity of edge server. The extensive experimental results demonstrate the ILSCS scheme can achieve lower the service cost and the service latency.

In our futher work, we will further investigate the caching problem of service instances with fault tolerance when some edge servers fail.

Authors' contributions

Binbin Huang, Ziqi Ran and Yuanyuan Xiang wrote the main manuscript text and prepared all figures - original draft. Dongjin Yu and Xiaoying Shi validated the manuscript. Zhongjin Li, Zhengqian Xu supervised to complete this work on time.

Funding

(details of any funding received). Zhejiang Provincial National Science Foundation of China, LY23F020015. National Natural Science Foundation of China, 61903109.

Availability of data and materials

(a statement on how any datasets used can be accessed). 1)The experiment data supporting this experiment analysis are from previously reported studies, which have been cited. 2)The experiment data used to support the findings of this study are included within the article. 3)The experiment data are described in Experimental evaluation in detail.

Declarations

Ethics approval and consent to particpate

(applicable for both human and/ or animal studies. Ethical committees, Internal Review Boards and guidelines followed must be named. When applicable, additional headings with statements on consent to participate and consent to publish are also required). This declaration is not applicable.

Competing interests

The authors declare no competing interests.

Received: 29 July 2022 Accepted: 14 August 2023
Published online: 13 September 2023

References

1. Wang T, Mei Y, Jia W, Zheng X, Wang G, Xie M (2020) Edge-based differential privacy computing for sensor–cloud systems. J Parallel Distrib Comput 136:75–85. https://doi.org/10.1016/j.jpdc.2019.10.009
2. J. Xu, L. Chen, and P. Zhou (2018) "Joint service caching and task offloading for mobile edge computing in dense networks," arXiv
3. S. Li, L. Da Xu, and S. Zhao (2018) "5G Internet of Things: A survey A R T I C L E I N F O," J. Ind. Inf. Integr. 10:1–9. Available: https://doi.org/10.1016/j.jii.2018.01.005
4. Zhang, Junna, Jiawei Chen, et al. (2023) "Dependent Task Offloading Mechanism for Cloud–Edge-Device Collaboration." J Netw Comput Appl. 216:103656, https://doi.org/10.1016/j.jnca.2023.103656
5. Zhang, Junna, Xiaoyan Zhao, et al. (2022) "A Composite Service Provisioning Mechanism in Edge Computing." Mobile Inf Syst. 2022:1–16, https://doi.org/10.1155/2022/9031201
6. Xia X, Chen F, He Q, Cui G, Lai P, Abdelrazek M, Grundy J, Jin H (2020) Graph-based data caching optimization for edge computing. Future Gener Syst 113:228–239. https://doi.org/10.1016/j.future.2020.07.016
7. Xia X, Chen F, He Q, Grundy J, Abdelrazek M, Jin H (2020) Online collaborative data caching in edge computing. IEEE Trans Parallel Distrib Syst 32(2):281–294. https://doi.org/10.1109/TPDS.2020.3016344
8. B. Huang, X. Liu, Y. Xiang, D. Yu, S. Deng and S. Wang, (2022) "Reinforcement learning for cost-effective IoT service caching at the edge. "J Parallel Distributed Comput. 168 https://doi.org/10.1016/j.jpdc.2022.06.008
9. Xia X, Chen F, Grundy J, Abdelrazek M, Jin H, He Q (2022) Constrained app data caching over edge server graphs in edge computing environment. IEEE Trans Serv Comput 15(5):2635–2647. https://doi.org/10.1109/TSC.2021.3062017
10. Du B (2021) Mobile edge computation induced caching strategy for huge online education with college teachers and students. Internet Technol Lett 4(1):2–7. https://doi.org/10.1002/itl2.208
11. Rim M, Kang CG (2020) Content Prefetching of Mobile Caching Devices in Cooperative D2D Communication Systems. IEEE Access 8:141331–141341. https://doi.org/10.1109/ACCESS.2020.3012442
12. Qi K, Han S, Yang C (2019) Learning a Hybrid Proactive and Reactive Caching Policy in Wireless Edge under Dynamic Popularity. IEEE Access 7:120788–120801. https://doi.org/10.1109/ACCESS.2019.2936866
13. Wang W, Lan RN, Gu JX, et al (2017) Edge caching at base stations with device-to-device offloading. IEEE Access 5:6399–6410. https://doi.org/10.1109/ACCESS.2017.2679198
14. Ahani G, Yuan D (2020) "Optimal scheduling of content caching subject to deadline." arXiv. 1:293–307. https://doi.org/10.1109/ojcoms.2020.2978585
15. C. K. Kim, T. Kim, A. Cho, and S. K. Lee (2020) "Delay-aware distributed caching scheme in edge network," Conex. 2020 - Proc. 16th Int. Conf. Emerg. Netw. Exp. Technol. 544–545, https://doi.org/10.1145/3386367.3431664
16. Gu H, Wang H (2020) A Distributed Caching Scheme Using Non-Cooperative Game for Mobile Edge Networks. IEEE Access 8:142747–142757. https://doi.org/10.1109/ACCESS.2020.3009683
17. M. Kim, H. Cho, Y. Cui, and J. Lee (2020) "Service Caching and Computation Resource Allocation for Large-Scale Edge Computing-Enabled Networks," 2020 IEEE Glob. Commun. Conf. GLOBECOM 2020 - Proc., https://doi.org/10.1109/GLOBECOM42002.2020.9322297
18. Ren Y (2021) Game theory based cooperative caching strategy in information-centric networking. Internet Technol Lett 4(1):2–5. https://doi.org/10.1002/itl2.160
19. Song, Jiongjiong, et al. (2017) "Learning Based Content Caching and Sharing for Wireless Networks." IEEE Transactions on Communications. 1–1, https://doi.org/10.1109/tcomm.2017.2713384
20. Lu S et al (2022) A Dynamic Service Placement Based on Deep Reinforcement Learning in Mobile Edge Computing. Network. 2:106–122. https://doi.org/10.3390/network2010008
21. Chen Y et al (2023) A Distributed Game Theoretical Approach for Credibility-Guaranteed Multimedia Data Offloading in MEC. Inf Sci. 644:119306. https://doi.org/10.1016/j.ins.2023.119306
22. Liang J, Ma B, Feng Z, Huang J (2023) Reliability-aware Task Processing and Offloading for Data-intensive Applications in Edge Computing. IEEE Trans Netw Serv Manage. https://doi.org/10.1109/TNSM.2023.3258191
23. Chen Y, Hu J, Zhao J, Min G. QoS-Aware Computation Offloading in LEO Satellite Edge Computing for IoT: A Game-Theoretical Approach[J]. Chinese Journal of Electronics. https://doi.org/10.23919/cje.2022.00.412
24. Chen, Ying, Jie Zhao, Jintao Hu, et al. (2023) "Distributed Task Offloading and Resource Purchasing in Noma-Enabled Mobile Edge Computing: Hierarchical Game Theoretical Approaches." ACM Transactions on Embedded Computing Systems. https://doi.org/10.1145/3597023
25. Zhang J, Chen D, Yang Q et al (2023) Proximity Ranking-Based Multimodal Differential Evolution. Swarm and Evolutionary Computation. 78:101277. https://doi.org/10.1016/j.swevo.2023.101277
26. Z Xu, L Zhou, S Chi-Kin Chau, W Liang, Q Xia, P Zhou (2020) "Collaborate or Separate? Distributed Service Caching in Mobile Edge Clouds." IEEE INFOCOM. 2020-July, 3:2066–2075. https://doi.org/10.1109/INFOCOM41043.2020.9155365
27. Zhang X et al (2022) Joint Edge Server Placement and Service Placement in Mobile-Edge Computing. IEEE Internet of Things Journal. 9:11261–11274. https://doi.org/10.1109/jiot.2021.3125957
28. Y. Chen, W. Gu, J. Xu, Y. Zhang and G. Min, "Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning," in China Communications, https://doi.org/10.23919/JCC.ea.2022-0372.202302.
29. Liu F, Huang J, Wang X (2023) Joint Task Offloading and Resource Allocation for Device-Edge-Cloud Collaboration with Subtask Dependencies. IEEE Transactions on Cloud Computing. https://doi.org/10.1109/TCC.2023.3251561

30. Huang J, Wan J, Lv B, Ye Q, Chen Y (2023) Joint Computation Offloading and Resource Allocation for Edge-Cloud Collaboration in Internet of Vehicles via Deep Reinforcement Learning. IEEE Syst J 17(2):2500–2511. https://doi.org/10.1109/JSYST.2023.3249217
31. Y. Chen, J. Zhao, Y. Wu, J. Huang and X. S. Shen, (2022) "QoE-Aware Decentralized Task Offloading and Resource Allocation for End-Edge-Cloud Systems: A Game-Theoretical Approach," in IEEE Transactions on Mobile Computing. https://doi.org/10.1109/TMC.2022.3223119
32. Jiang W, Feng G, Qin S, Yum TSP, Cao G (2019) Multi-Agent Reinforcement Learning for Efficient Content Caching in Mobile D2D Networks. IEEE Trans Wirel Commun 18(3):1610–1622. https://doi.org/10.1109/TWC.2019.2894403
33. Kapetanakis S, Kudenko D, Strens MJA (2003) Reinforcement learning approaches to coordination in cooperative multi-agent systems. Lect. Notes Artif. Intell. (Subseries Lect. Notes Comput. Sci. 2636:18–32. https://doi.org/10.1007/3-540-44826-8_2
34. Korbut AA, Sigal IK (2010) Exact and greedy solutions of the knapsack problem: The ratio of values of objective functions. J Comput Syst Sci Int 49(5):757–764. https://doi.org/10.1134/S1064230710050102

Publisher's Note