An Online Machine Learning-based Content Caching Scheme in Mobile Edge Computing Networks

Qi Zhao[a], Yi Li[a], Hang Liu[b], Nicholas DeCortec[c], Frank Tucker[c], and Genshe Chen[a]

[a]Intelligent Fusion Technology, Inc., Germantown, MD, USA
[b]The Catholic University of America, Washington, DC, USA
[c]U.S. Army Combat Capabilities Development, Orlando, FL, USA

ABSTRACT

Mobile Edge Computing (MEC) is an emerging and fast-growing distributed computing paradigm. It brings the computation and storage resources closer to mobile users while also processing data at the network edge to improve response time and save bandwidth. In tactical virtual training environments, latency is a key factor that affects training performance. Additionally, MEC provides both information service environment and cloud computing capabilities to enable real-time virtual training. Therefore, we designed a machine learning-based data caching and processing scheme for the virtual training networks. The design consists of three tiers, mobile devices, edge servers, and cloud servers, respectively. By pre-caching the critical content objects close to the mobile devices, our MEC network enables data transmission and processing at low latency. Utilizing machine learning techniques, our caching scheme can predict and select the content objects to be cached with optimal storage efficiency at network edge servers. Specifically, we decoupled the content caching problem into two subproblems, namely probability learning and content selection. For probability learning, the edge servers estimate the probability and frequency that each content object will be requested in the near future. The estimate is according to the content request pattern learned over time. For the content selection, the edge servers determine the content objects for caching to minimize the expected content delay with limited storage. To evaluate the performance of our proposed scheme, we developed a testbed with real mobile devices and servers. The experimental results validated the feasibility and significant performance gains of the proposed scheme.

Keywords: Content caching, latency reduction, machine learning, mobile edge computing

1. INTRODUCTION

Edge computing refers to enabling technologies that allow computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of Internet of Things (IoT) services.[1] The term "edge" refers to any computing or network resources on the path between data sources and cloud data centers. In most cases, edge devices are closer to data sources. For example, a smartphone can be the edge device between the user and cloud data centers, and a small gateway server can be the edge device between User Devices (UEs) and cloud data centers. Therefore, the computation and data storage can happen at the location where it is needed. This creates the opportunity for the responding time of data requesting to be reduced, while also allowing network resources, such as bandwidth, to be saved. Previous works[2,3] provide the evidence that the edge computing paradigm can efficiently reduce the data response time to improve the performance of the applications running on the UE. The aim of edge computing is to move the computation away from data centers and towards the edge of the network. Thereby, exploiting smaller units in the network to perform tasks and provide services on behalf of the cloud data centers. The ultimate goal of edge computing is to provide content caching, service delivery, data storage and device management resulting in better data response time and data transfer rate. Globally, edge computing is rapidly progressing, and in recent years it has proven to dramatically increase its impact.

In 2014, the European Telecommunications Standard Institute (ETSI) proposed a brand new concept of Mobile Edge Computing (MEC).[4] It defines a new computing and networking platform that provides IT and cloud-computing capabilities within the Radio Access Network (RAN) in close proximity to mobile subscribers.[4] The original concept of MEC is to offload computation tasks of mobile devices to Base Stations (BSs). The

Sensors and Systems for Space Applications XV, edited by Genshe Chen, Khanh D. Pham, Proc. of SPIE
Vol. 12121, 121210D · © 2022 SPIE · 0277-786X · doi: 10.1117/12.2623040

development of MEC and cellular networks provide edge devices a much broader range, including smartphones, BSs, gateway servers and set-top boxes.[5] Unlike before, MEC can now provide better computing, networking performance and Quality of Service (QoS) in terms of higher bandwidth and lower latency.[6] Another recent emerged technology concept is called fog computing,[5] which has overlaps with MEC. In the current 5G, even 6G, development trend, both MEC and fog computing play an important role. Both are able to provide better computing and networking efficiency to meet the high standards and demands.[7]

However, MEC brings in new challenges as well. Due to the rapidly increasing speed of the total amount of mobile contents generated by mobile devices, the network is overwhelmed. The vast amount of transmitting and processing contents within the network presents a huge burden. Thus, mobile content should be delivered to the correct place without data lost or quickly processed by the appropriate entity in the network within a certain time period. On the other hand, the total amount of contents in the network is increasing fast as well. There should be a vast amount of redundant contents which can be eliminated to save the network bandwidth and improve the content delivery performance. Therefore, content caching scheme[8] has emerged as a promising approach to yield significant improvements for the content transmission and delivery. In general, popular or frequently requested contents can be cached in the network close to the end devices by utilizing the storage resources at the edge devices. In this way, multiple benefits can be achieved for a better networking environment in the MEC context. For instance, the end-user devices can obtain the desired contents much faster. Additionally, the volume of the total network traffic can be significantly reduced as well as an improvement in the efficiency and utilization of the resources in the network (such as network bandwidth and network storage).

There are a lot of existing works focusing on how to make the caching solution better as introduced in the following section. Common problems often studied are which content to cache, where to cache, and how to cache. In this paper, we mainly focus on which content to cache and where to cache in the MEC environment. Specifically, we propose a dynamic real-time machine learning-based content caching scheme for distributed cloud data centers and edge servers to retrieve, cache, distribute and collect training data to optimize the quality of service to mobile devices. Our content caching scheme will intelligently decide the popularity of the contents requested by end users via learning from the past, and then select the optimal edge device to cache the content. Additionally, we developed a real testbed with real servers and Kernal-based Virtual Machines (KVM)[9] Virtual Machines (VMs) for validating and evaluating our solution. We also designed different testing scenarios based on the tactical communication environment to emulate real MEC environment. Through the evaluation tests, our proposed solution promises to provide optimal content caching strategy while meeting the QoS requirements (i.e., latency) with minimal effort. Our main contributions in this work are as follows:

1. We proposed a machine learning-based content caching scheme for dynamically caching the most popular content in the edge devices in MEC environment.
2. We divided the content caching problem into two subproblems (content popularity subproblem and content storage selection subproblem) and solved them individually.
3. We built a real testbed with real servers and emulated networking environment to validate and evaluate our proposed content caching scheme.
4. We conducted different testings to validate our proposed scheme and completely evaluate its performance. Based on the evaluation results, our proposed scheme can provide promising caching strategies and the benefits that we claimed can be achieved as well.

In addition, there are five other related areas addressed. Discussed in Section 2 is the recent work related to content caching and MEC. Introduced in Section 3 is the system model and problem formulation of the machine learning-based content caching and processing scheme. Presented in Section 4 is the design of the machine learning-based content caching algorithm. Described in Section 5 is the numerical evaluation results of the simulation experiment conducted on our emulated network testbed, along with the validation and analysis. Lastly, Section 6 draws the conclusion for our work.

2. RELATED WORK

In recent years, due to the rapid increase in the total volume of network traffic, the MEC paradigm and content caching have emerged as promising approaches. Additionally, the QoS requirements of mobile end users, such as delay and bandwidth, is becoming progressively demanding, particularly since entering into the 5G era. Several studies and works are attempting to bring more efficient and intelligent networking mechanisms to further explore the potential of MEC and content caching. The target is to yield significant gains for both end-users and network operators.

The MEC paradigm has emerged as a key enabling technology for realizing the IoT and 5G visions.[4,6,10,11] Resource management is a hot topic in MEC. For example, general guidelines were designed[12,13] to determining the optimal offloading decision for the purpose of minimizing the mobile-energy consumption and computation latency. Another work[14] formulated the problem of transmission-energy minimization under a computation-deadline constraint with the optimization variable being the input-data transmission time. At which point the famous Shannon-Hartley formula gives the power-rate function. One year before that, another study[15] was conducted to minimize the energy consumption for executing a task with a soft real-time requirement, targeting, e.g., multimedia applications. This required the task to be completed within a deadline of a given probability. In our previous work,[16,17] we also designed a task offloading scheme in the MEC environment to handle the task distribution, offloading and management by applying deep reinforcement learning. Specifically, we formulated the task offloading problem as a multi-agent reinforcement learning problem. The decision-making process of each agent was modeled as a Markov decision process. The deep Q-learning approach was applied to deal with the large scale of states and actions. Later on, inspired by the parallel computing technique, a new solution leveraging the partial offloading (also known as program partitioning) was proposed[18–25] to further optimize MEC performance.

In 2018, a new cooperative edge caching architecture[26] for 5G networks, where mobile edge computing resources are utilized for enhancing edge caching capability, was proposed. In the architecture, they focused on mobility-aware hierarchical caching, where smart vehicles are taken as collaborative caching agents for sharing content cache tasks with base stations. To further utilize the caching resource of smart vehicles, they also introduced a new vehicular caching cloud concept, and proposed a vehicle-aided edge caching scheme, where the caching and computing resources at the wireless network edge are jointly scheduled. Another study[27] proposed a proactive caching mechanism named learning-based cooperative caching strategy. This strategy was based on MEC architecture to reduce transmission cost while improving user quality of experience for future mobile networks. In that architecture, they exploited a transfer learning-based approach for estimating content popularity and then formulate the proactive caching optimization model. In that year, a novel MEC-enabled wireless blockchain framework[28] was presented. This framework proposed that the computation-intensive mining tasks can be offloaded to nearby edge computing nodes. It also proposed that the cryptographic hashes of blocks can be cached in the MEC server. Particularly, two offloading modes were considered, i.e., offloaded to the nearby access point or a group of nearby users. They conducted the performance analysis of each mode with stochastic geometry methods, and the joint offloading decision and caching strategy was formulated as an optimization problem. Similarly, a learning-based cooperative content caching policy was proposed[29] for the MEC architecture which addressed unknown user preference and only observed historical content demands can be proposing. They modeled the cooperative content caching problem as a multi-agent and multi-armed bandit problem, proposing a multiagent reinforcement learning-based algorithm to solve the problem. Inclusive, network coding was proposed to be leveraged for making content caching strategies,[30] and there are still several other works[31–36] presented that to address content caching in MEC and mobile wireless networks.

3. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we describe our system model and the current content caching problem. A detailed problem formulation is also provided to support a quantitative analysis. Based on the problem formulation, an efficient machine learning based content caching algorithm is designed. This will be introduced in Section 4.

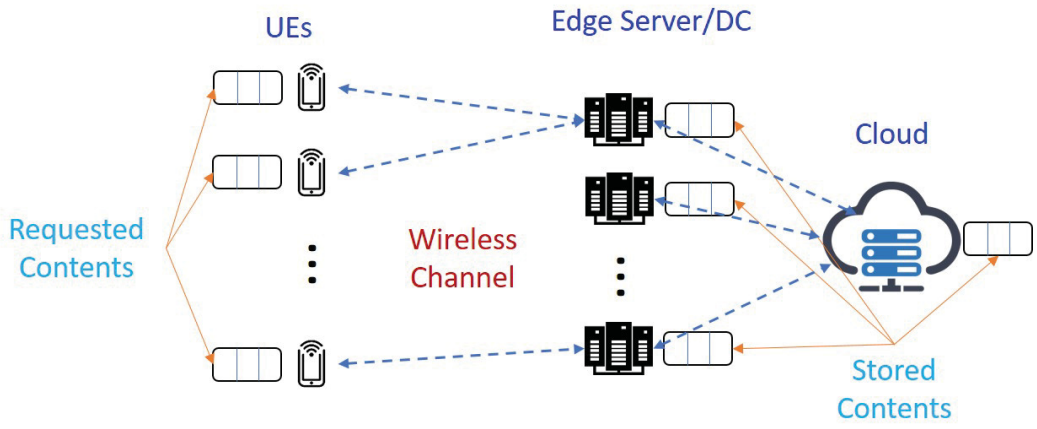

Figure 1. System Model

3.1 System Model and Scenario

As shown in Fig.1, we consider a content delivery system with J data centers (DCs), indexed by $j \in \{1, \cdots, J\}$. This provides content delivery services to the K mobile user equipment (UEs) indexed by $k \in \{1, \cdots, K\}$. Each data center is located at an edge server. Thus, we use the same index for both DC and edge server. We also assume that each UE is associated to one edge server at each time. The content placement is updated periodically, and we index these periods by $t = 1, \cdots, T$. Each UE randomly requests any one of the N possible contents, which are indexed by $i \in \{1, \cdots, N\}$. Denote $\theta_{i,k}$ as the popularity of content i to UE k, which is defined as the probability that content i is requested by UE k. Hence, we have $\sum_{i=1}^{N} \sum_{k=1}^{K} \theta_{i,k} = 1$. The content popularity file over K UEs and N contents can be denoted by the matrix below, which is assumed to be unknown without a learning process.

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_{1,1} & \cdots & \theta_{1,K} \\ \vdots & \ddots & \vdots \\ \theta_{N,1} & \cdots & \theta_{N,K} \end{bmatrix}, \tag{1}$$

Each DC has a limited storage space that stores the files of a certain set of contents. Associated UEs are allowed to download these contents directly from the DC, without having to download them from the remote cloud server via the Internet. In each period t, the content placement strategy of DC j is indicated by a set of binary variables, given by

$$a_{i,j}^{[t]} = \begin{cases} 1, & \text{if content } i \text{ is stored at DC } j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

for $i \in \{1, \cdots, N\}, j \in \{1, \cdots, J\}$, and $t \in \{1, \cdots, T\}$.

Let s_i be the size of content i (in bytes) and let Z_j be the storage space of DC j (in bytes). Then, for all DCs $j = 1, \cdots, J$, $a_{i,j}^{[t]}$ should satisfy

$$\sum_{i=1}^{N} a_{i,j}^{[t]} s_i \leq Z_j \tag{3}$$

for $j \in \{1, \cdots, J\}$ and $t \in \{1, \cdots, T\}$.

When the contents at a DC need to be updated, the DC will download the contents to be added from the cloud server via backhaul and deletes the contents to be removed. Each UE is associated with one DC at an edge server in our scenario. The association between DCs and UEs is indicated by a set of $0-1$ variables given by

$$x_{k,j}^{[t]} = \begin{cases} 1, & \text{if UE } k \text{ is stored at DC } j \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

for $k \in \{1, \cdots, K\}, j \in \{1, \cdots, J\}$, and $t \in \{1, \cdots, T\}$. Suppose UE k is associated with DC j at time t (i.e., $x_{k,j}^{[t]} = 1$) and it requests content i, the content will be directly sent by DC j to UE k with low latency. However, this occurs only if content i is stored at DC j. Otherwise, UE k can only download the content from the cloud server via Internet, which will take a much longer time.

The content placement strategy of each DC over $t = 1, \cdots, T$ is called a policy, denoted by π_j $(j = 1, \cdots, J)$. The joint policy of all DCs is denoted by $\boldsymbol{\pi} = \{\pi_1, \cdots, \pi_J\}$. Obviously, the latency performance is dictated by $\boldsymbol{\pi}$.

3.2 Problem Formulation

We assume that the communication between DCs and UEs is supported by a wireless network based on orthogonal frequency-division multiple access (OFDMA). Denote the average data rate assigned to UE k in the period t when associated with DC j as $R_{k,j}$, which can be evaluated through measurement. Suppose UE k is associated with DC j and it requests content i at time t. If content i is stored at DC j (i.e., $a_{i,j}^{[t]} = 1$), the latency experienced by UE k is the downloading time of content i from DC j to UE k, which is given by

$$E_{k,j,i}^{[t]} = s_i / R_{k,j}^{[t]}. \tag{5}$$

If content i is not stored at DC j (i.e., $a_{i,j}^{[t]} = 0$), UE k must download the content from the cloud server via the Internet. It should be noted that the physical connection between a UE and the cloud server may span multi-hop wireless and wired links, each with its own traffic dynamics and medium access mechanisms. Thus, the end-to-end communication path can be highly dynamic, and we use the time-averaged downloading time of each content in our analysis. Let $F_{k,i}$ be the average downloading time of content i for UE k. Considering both cases (i.e., content i available or unavailable), the latency of UE k requesting content i when associated with DC j is given by

$$D_{k,j,i}^{[t]} = a_{i,j}^{[t]} E_{k,j,i}^{[t]} + (1 - a_{i,j}^{[t]}) F_{k,i}. \tag{6}$$

We assume that each UE is constantly associated with only one DC in a period, then we can simply denote the latency of UE k requesting content i as

$$D_{k,i}^{[t]} = D_{k,j,i}^{[t]}. \tag{7}$$

Since the UE k may request any one of the N contents with probabilities $\{\theta_{i,k}\}, i = 1, \cdots, N$, the expected latency of UE k at time t is calculated by

$$D_k^{[t]} = \frac{\sum_i \theta_{i,k} D_{k,i}^{[t]}}{\sum_i \theta_{i,k}}. \tag{8}$$

In this work, we aim to find the optimal joint policy of all DCs $\boldsymbol{\pi}$ (i.e., the sequence of program placement strategies of all DCs over $t = 1, \cdots, T$) that minimizes the average latency of all UEs, given by $\sum_{k=1}^{K} \left(\sum_i \theta_{i,k}\right) D_k^{[t]}$. Such a problem is formulated as

$$\mathbf{P1}: \quad \min_{\pi} \sum_{k=1}^{K} \left(\sum_i \theta_{i,k} \right) D_k^{[t]} \tag{9}$$

$$\mathbf{s.t.} \quad \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \le Z_j, \quad j = 1, \cdots, J \tag{10}$$

$$a_{i,j}^{[t]} \in \{0, 1\}, \quad i = 1, \cdots, N \quad j = 1, \cdots, J. \tag{11}$$

In **P1**, the first constraint is due to the storage space limit of each DC. In the next section, we will provide our own solution to find the optimal content caching policy.

4. MACHINE LEARNING BASED CONTENT CACHING ALGORITHM

In this section, an efficient solution for the problem **P1** is proposed. All the decision variables $\{a_{i,j}^{[t]}\}$ in **P1** are $0-1$ integers. To derive a low-complexity solution algorithm, we decompose the original problem **P1** into two subproblems to be solved in each period. The first subproblem is content popularity acquisition, which aims to obtain the probability that each content will be requested by each UE, i.e., $\{\theta_{i,k}\}$. The second subproblem is content placement, in which each DC determines which content should be cached in its local storage.

4.1 Content Popularity Profile Acquisition

The content popularity can be estimated independently at each edge server according to the requests it receives from the UEs. The edge server can update the estimated popularity matrix θ periodically via exponential averaging, given as

$$\theta_\tau = \alpha\theta_{\tau-\mathbf{1}} + (1-\alpha)\hat{\theta_\tau} \tag{12}$$

where τ is the iteration index for content popularity updating, factor $\alpha \in [0,1)$, and $\hat{\theta_\tau}$ denotes the popularity matrix estimated only using the requests received during the τ-th iteration. Note that the popularity of content i to UE k during the τ-th iteration is approximated as the number of requests from UE k for content i over the total number of received requests.

Since each UE is only associated with one edge server, each edge server only receives requests from a subset of the UEs. Hence, each edge server can only estimate part of the popularity matrix. However, each edge server only needs to satisfy the associated UEs. It is sufficient for the edge server to perform content placement with partial popularity knowledge.

For simplicity, the updating content popularity can be performed immediately before updating the content placement policy, at the beginning of each time period t. We summarize our content popularity profile acquisition algorithm in **Algorithm** 1. In this algorithm, the current popularity matrix is denoted as θ, the request counting matrix is presented as Req. Thus, $Req[i,k]$ represents the number of requests received from the k-th user for the i-th content in the most recent time frame, and function $Zeros(N,K)$ returns a $N \times K$ zero matrix.

Algorithm 1 Content Popularity Profile Acquisition

```
θ ← Zeros(N, K)
Req ← Zeros(N, K)
while True do
    while The current time frame is not over do
        Wait for a request from UEs
        Determine the UE index k and requested content index i
        Req[i, k] ← Req[i, k] + 1
    end while
    ReqAll ← Σ_i Σ_k Req[i, k]
    θ̂ ← Req/ReqAll
    if This is the first time frame then
        θ ← θ̂
    else
        θ ← αθ + (1 − α)θ̂
    end if
    Req ← Zeros(N, K)
end while
```

4.2 Content Placement with Given User Association

With an updated content popularity profile θ at the beginning of a time frame, we then optimize the content placement strategy at each edge server. Denote $\mathbf{U_j}$ as the set of UEs associated with the j-th edge server, and

$\mathbf{C_j}$ as the set of contents cached at the j-th DC. The average content transmission delay is

$$D^{avg} = \sum_{k=1}^{K} \sum_{i} \theta_{i,k} D_{k,i}^{[t]}. \tag{13}$$

Thus, when caching content i at edge server j, the average delay reduction for UE k is given by

$$\Delta D_{k,i}^{[t]} = \theta_{i,k} \left(F_{k,i} - E_{k,j,i}^{[t]} \right). \tag{14}$$

Thus, minimizing the average delay at DC j, denoted by D_j^{avg}, is equivalent to maximizing $\sum_{k \in \mathbf{U_j}} \sum_{i \in \mathbf{C_j}} \Delta D_{k,i}^{[t]}$ under the storage size constraint at the edge server j. Then, the content placement problem at the edge server j can be formulated as

$$\mathbf{P2}: \quad \max_{\left\{a_{i,j}^{[t]}\right\}} \sum_{k \in \mathbf{U_j}} \sum_{i \in \mathbf{C_j}} \Delta D_{k,i}^{[t]} \tag{15}$$

$$\mathbf{s.t.} \quad \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \leq Z_j, \quad j = 1, \cdots, J \tag{16}$$

$$a_{i,j}^{[t]} \in \{0, 1\}, \quad i = 1, \cdots, N \quad j = 1, \cdots, J. \tag{17}$$

The program placement optimization (i.e., **P2**) is a Knapsack problem, which is generally NP-hard. Since this problem has to be solved in each period, it is necessary to develop a low-complexity solution that provides timely outcomes. The greedy algorithm is a common approach to solving a Knapsack problem and can be applied to obtain an efficient solution. Specifically, the proposed greedy content placement strategy allocates the storage space of each DC to the contents with the highest delay reduction per occupied space, which is $\sum_{k \in \mathbf{U_j}} \Delta D_{k,i}^{[t]} / s_i$. To implement the greedy algorithm, we first sort all contents by the descending order of $\sum_{k \in \mathbf{U_j}} \Delta D_{k,i}^{[t]} / s_i$. Then, the contents are put into the storage space of DC j, following such order until no more content can be further stored in the storage of DC j. The proposed greedy content placement strategy for DC j can be summarized as the **Algorithm** 2 below.

Algorithm 2 Content Placement

Set all $a_{i,j}^{[t]} \leftarrow 0$ for $i = 1, \cdots, N$
Set $\mathbf{U_j} = \phi$
Compute $G_i \leftarrow \sum_{k \in \mathbf{U_j}} \Delta D_{k,i}^{[t]} / s_i$ for $i = 1, \cdots, N$
while $Z_j - \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \geq \min_i \left\{ s_i | a_{i,j}^{[t]} = 0 \right\}$ **do**
  $i^* \leftarrow \arg\max_i \left\{ G_i | a_{i,j}^{[t]} = 0 \right\}$
  if $Z_j - \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \geq s_{i^*}$ **then**
    $a_{i^*,j}^{[t]} \leftarrow 1$
  end if
  $G_{i^*} \leftarrow -1$
end while

5. NUMERICAL RESULTS

We performed several evaluation experiments on the content placement scheme to validate the feasibility of our proposed solution and evaluate the performance. Specifically, we conducted three different experiments to test in our testbed. As shown in Figure 2, our testbed is implemented by using a local network consisting of three desktops. One desktop serves as the remote cloud server and the rest serve as the edge server. In order to

request contents, the user devices can access any one of the two edge servers through http protocol. We randomly generate 50 contents at the cloud server, and the size of each content is randomly selected from 0.1 MB to 5 MB. Poisson process is used to generate the content requests from the user device. The request intensity or request arrival rate is denoted as λ_{avg}. This indicates the average number of requests that will be generated for each mobile user in 1 second. In the testing results below, we investigate the impacts of each factor. The investigation includes the storage size at the edge server, the number of contents, and the request intensity on the average delay for the users to get the requested contents.

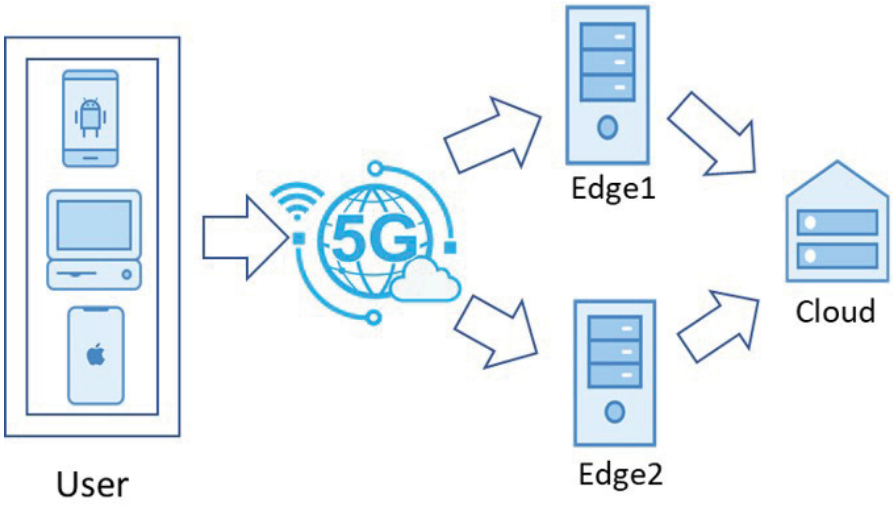

Figure 2. Testbed Architecture

In the first experiment, we investigate the impact of the storage size of the edge server. We gradually increase the storage size of the edge server from 0 MB to 150 MB, and plot the curve for the average delay in Figure 3. Intuitively, with more storage space, the edge server is able to cache more contents. As a result, a lower average delay is expected. When the storage size is 0 MB, the edge server can cache nothing locally and every requested content has to be fetched from the cloud server. This situation is equivalent to the case without content caching, in which we have the maximum average delay of around 4.75 seconds. As the storage size increases, the average delay reduces. When the storage size reaches 150 MB, all contents can be cached at the edge server. In this situation, we can fully avoid the transmission delay from the cloud to the edge. We can achieve a minimum average delay of around 0.13 seconds. From Figure 3, we also observe that the average delay drops fast at the beginning when increasing the storage size from 0 MB to 50 MB, and the delay reducing rate gets slower after that. Since our caching algorithm selects the caching contents according to their average delay reduction achieved per unit storage space, the content that can provide more delay reduction will be cached first. Therefore, our caching method can provide outstanding delay improvement even with limited storage space.

In the second experiment, we investigate the impact of the number of contents. In this experiment, we fix the storage size at the edge server at 50 MB, and change the number of contents in the DC. We randomly select a subset of contents from the entire 50 contents in the DC. Additionally, we restrict the mobile users to request only from the selected subset to perform the experiments. In the beginning, we select 10 contents and all mobile users request from the selected 10 contents. Since the number of content is small, most of the content can be cached by the edge server, and we can achieve a minimum average delay of around 0.16 seconds. As the number of contents increases, the ratio of the contents that can be cached keeps reducing. As shown in Figure 4, the average delay increases as the number of content increases.

In the third experiment, we use all 50 contents, set the storage size at the edge server to 50 MB, and change the request arrival rate λ_{avg} from 0.5 requests per second to 1.2 requests per second. Intuitively, as the request intensity increases, the traffic load increases for both the edge network and the backbone network between cloud

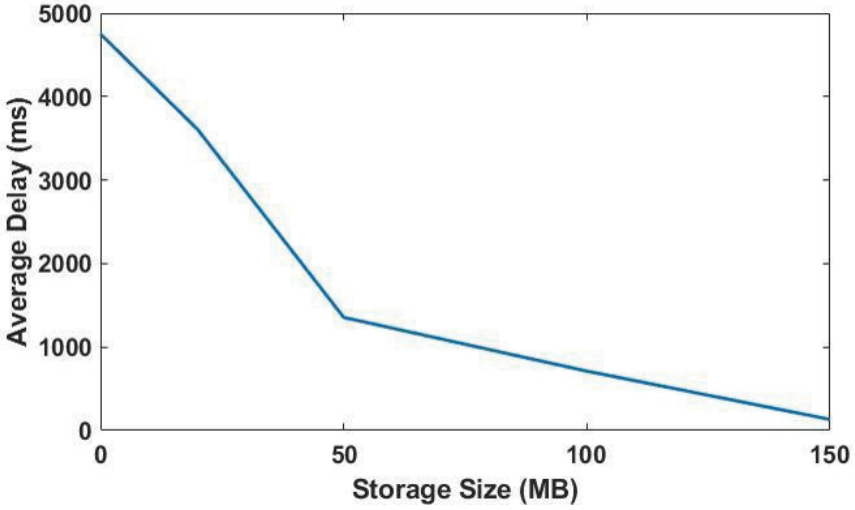

Figure 3. Content Cache Size vs. Average Content Request Delay

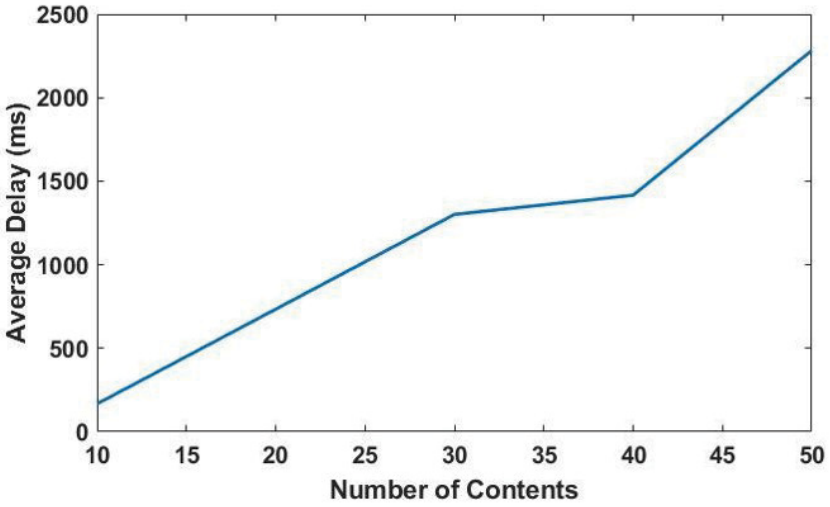

Figure 4. Number of Contents vs. Average Content Request Delay

and edge servers. As shown in Figure 5, the average delay increases as the request intensity increases. When the request arrival rate is 0.5 requests per second, we achieve a minimum average delay of around 1.03 seconds. When the request arrival rate reaches 1.2 requests per second, we reach a maximum average delay of around 6.18 seconds. When the request intensity is below a certain threshold, the network can handle the content delivery well, and the delay increment is not significant. After the content arrival rate exceeds a threshold, the network can no longer handle the content delivery and the average delay increases dramatically.

6. CONCLUSION

In this paper, we investigated the content caching problem for the mobile edge computing scenario, and we designed a solution for obtaining the optimal content caching allocation strategy. Specifically, we formulated the content caching problem as an average delay minimization problem with the storage constraint. Then, we proposed a machine learning-based algorithm to solve this content caching problem, which decouples it into two subproblems, i.e., content popularity acquisition and content placement. For each subproblem, we designed

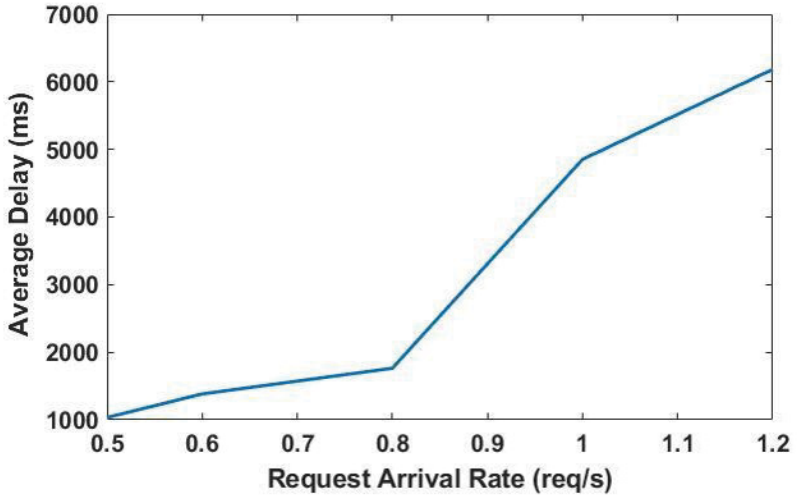

Figure 5. Request Arrival Rate vs. Average Content Request Delay

an efficient and effective algorithm to solve them respectively. In the end, we implemented our solution on our emulated network testbed. Through numerical evaluation results, we investigated different impacts of the storage size at the data center, the total content number, and the request rate of UEs.

REFERENCES

[1] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L., "Edge computing: Vision and challenges," *IEEE Internet of Things Journal* **3**(5), 637–646 (2016).

[2] Yi, S., Hao, Z., Qin, Z., and Li, Q., "Fog computing: Platform and applications," in [*2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*], 73–78 (2015).

[3] Ha, K., Chen, Z., Hu, W., Richter, W., Pillai, P., and Satyanarayanan, M., "Towards wearable cognitive assistance," in [*Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*], *MobiSys '14*, 68–81, Association for Computing Machinery, New York, NY, USA (2014).

[4] Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V., "Mobile edge computing—a key technology towards 5g," *ETSI white paper* **11**(11), 1–16 (2015).

[5] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S., "Fog computing and its role in the internet of things," in [*Proceedings of the first edition of the MCC workshop on Mobile cloud computing*], 13–16 (2012).

[6] Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B., "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials* **19**(4), 2322–2358 (2017).

[7] Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T., "Mobile edge computing: A survey," *IEEE Internet of Things Journal* **5**(1), 450–465 (2018).

[8] Safavat, S., Sapavath, N. N., and Rawat, D. B., "Recent advances in mobile edge computing and content caching," *Digital Communications and Networks* **6**(2), 189–194 (2020).

[9] Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A., "kvm: the linux virtual machine monitor," in [*Proceedings of the Linux symposium*], **1**(8), 225–230, Dttawa, Dntorio, Canada (2007).

[10] Shi, W. and Dustdar, S., "The promise of edge computing," *Computer* **49**(5), 78–81 (2016).

[11] Salman, O., Elhajj, I., Kayssi, A., and Chehab, A., "Edge computing enabling the internet of things," in [*2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*], 603–608, IEEE (2015).

[12] Kumar, K. and Lu, Y.-H., "Cloud computing for mobile users: Can offloading computation save energy?," *Computer* **43**(4), 51–56 (2010).

[13] Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B., "A survey of computation offloading for mobile systems," *Mobile networks and Applications* **18**(1), 129–140 (2013).

[14] Barbarossa, S., Sardellitti, S., and Di Lorenzo, P., "Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks," *IEEE Signal Processing Magazine* **31**(6), 45–55 (2014).

[15] Zhang, W., Wen, Y., Guan, K., Kilper, D., Luo, H., and Wu, D. O., "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications* **12**(9), 4569–4581 (2013).

[16] Zhao, Q., Feng, M., Li, L., Li, Y., Liu, H., and Chen, G., "Deep reinforcement learning based task scheduling scheme in mobile edge computing network," in [*Sensors and Systems for Space Applications XIV*], **11755**, 117550K, International Society for Optics and Photonics (2021).

[17] Feng, M., Zhao, Q., Sullivan, N., Chen, G., Pham, K., and Blasch, E., "Task assignment in mobile edge computing networks: a deep reinforcement learning approach," in [*Sensors and Systems for Space Applications XIV*], **11755**, 117550L, International Society for Optics and Photonics (2021).

[18] Jia, M., Cao, J., and Yang, L., "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in [*2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*], 352–357, IEEE (2014).

[19] Mahmoodi, S. E., Uma, R., and Subbalakshmi, K., "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Transactions on Cloud Computing* **7**(2), 301–313 (2016).

[20] Wang, Y., Sheng, M., Wang, X., Wang, L., and Li, J., "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications* **64**(10), 4268–4282 (2016).

[21] Kao, Y.-H., Krishnamachari, B., Ra, M.-R., and Bai, F., "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing* **16**(11), 3056–3069 (2017).

[22] Zhang, W., Wen, Y., and Wu, D. O., "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Transactions on Wireless Communications* **14**(1), 81–93 (2014).

[23] Khalili, S. and Simeone, O., "Inter-layer per-mobile optimization of cloud mobile computing: a message-passing approach," *Transactions on Emerging Telecommunications Technologies* **27**(6), 814–827 (2016).

[24] Di Lorenzo, P., Barbarossa, S., and Sardellitti, S., "Joint optimization of radio resources and code partitioning in mobile edge computing," *arXiv preprint arXiv:1307.3835* (2013).

[25] Mahmoodi, S. E., Subbalakshmi, K., and Sagar, V., "Cloud offloading for multi-radio enabled mobile devices," in [*2015 IEEE international conference on communications (ICC)*], 5473–5478, IEEE (2015).

[26] Zhang, K., Leng, S., He, Y., Maharjan, S., and Zhang, Y., "Cooperative content caching in 5g networks with mobile edge computing," *IEEE Wireless Communications* **25**(3), 80–87 (2018).

[27] Hou, T., Feng, G., Qin, S., and Jiang, W., "Proactive content caching by exploiting transfer learning for mobile edge computing," *International Journal of Communication Systems* **31**(11), e3706 (2018).

[28] Liu, M., Yu, F. R., Teng, Y., Leung, V. C., and Song, M., "Computation offloading and content caching in wireless blockchain networks with mobile edge computing," *IEEE Transactions on Vehicular Technology* **67**(11), 11008–11021 (2018).

[29] Jiang, W., Feng, G., Qin, S., and Liang, Y.-C., "Learning-based cooperative content caching policy for mobile edge computing," in [*ICC 2019-2019 IEEE International Conference on Communications (ICC)*], 1–6, IEEE (2019).

[30] He, W., Su, Y., Xu, X., Luo, Z., Huang, L., and Du, X., "Cooperative content caching for mobile edge computing with network coding," *IEEE Access* **7**, 67695–67707 (2019).

[31] Wang, C., Liang, C., Yu, F. R., Chen, Q., and Tang, L., "Joint computation offloading, resource allocation and content caching in cellular networks with mobile edge computing," in [*2017 IEEE international conference on communications (ICC)*], 1–6, IEEE (2017).

[32] Yuan, P., Cai, Y., Liu, Y., Zhang, J., Wang, Y., and Zhao, X., "Prorec: a unified content caching and replacement framework for mobile edge computing," *Wireless Networks* **26**(4), 2929–2941 (2020).

[33] Asheralieva, A., "Optimal computational offloading and content caching in wireless heterogeneous mobile edge computing systems with hopfield neural networks," *IEEE Transactions on Emerging Topics in Computational Intelligence* **5**(3), 407–425 (2019).

[34] Wang, N., Shen, G., Bose, S. K., and Shao, W., "Zone-based cooperative content caching and delivery for radio access network with mobile edge computing," *IEEE Access* **7**, 4031–4044 (2018).

[35] Yu, Z., Hu, J., Min, G., Lu, H., Zhao, Z., Wang, H., and Georgalas, N., "Federated learning based proactive content caching in edge computing," in [*2018 IEEE Global Communications Conference (GLOBECOM)*], 1–6, IEEE (2018).

[36] Zhang, M., Wang, S., and Gao, Q., "A joint optimization scheme of content caching and resource allocation for internet of vehicles in mobile edge computing," *Journal of Cloud Computing* **9**(1), 1–12 (2020).