Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 3501329, 11 pages
https://doi.org/10.1155/2022/3501329

Research Article

A New Heuristic Computation Offloading Method Based on Cache-Assisted Model

Junhua Wu[1], **Cang Fan**[1], **Guangshun Li**[1], **Zhuqing Xu**[1], **Zhenyu Jin**[1] and **Yuanwang Zheng**[2]

[1]*School of Computer Science, Qufu Normal University, Rizhao 276826, China*
[2]*Shandong Huatong Used Car Information Technology Limited Company, Jining 272000, China*

Correspondence should be addressed to Guangshun Li; guangshunli@qfnu.edu.cn

Received 21 January 2022; Revised 24 February 2022; Accepted 1 March 2022; Published 25 March 2022

Academic Editor: Yan Huo

Mobile edge computing (MEC) solves the high latency problem of cloud computing by offloading tasks to edge servers. Due to limited resources, it is necessary to improve the efficiency of computation offloading. However, there is a lot of redundant data transmission between MEC servers and users in the existing methods. Additional data transmission increases the task processing delay. To reduce the total delay, a new cache-assisted computation offloading strategy is proposed. In response to a large number of similar requests from users, a new cache management mechanism is designed. This mechanism can select reusable calculation results more accurately in the cache space through an approximate matching method and improve the cache hit ratio. Then, aiming at the problem of offloading efficiency, the delay optimization problem is transformed into an optimal path problem, a cost function is defined to determine the optimal offloading position, and an improved path planning method is used to plan the optimal offloading path. The simulation results indicate that the proposed scheme can improve the cache hit ratio and reduce the total processing delay of tasks compared with other standard schemes.

1. Introduction

With the development of Internet, increase in the number of network devices generates huge data traffic [1]. It is expected that in the next few years, the load on cloud data centers will be under tremendous pressure. Mobile edge computing (MEC) [2] as a new computing paradigm has been proposed to deal with the problem. MEC servers are deployed at the base stations (BSs) in the MEC system, which can execute the delay sensitive applications in close proximity to the end-users [3]. The edge servers can deploy the computation and storage resources to nearby IoT devices and offer data processing services [4]. Unlike traditional mobile cloud computing (MCC) [5], MEC is able to extend the cloud computing power and the services to the edge of network for two reasons: on the one hand, MEC ensures that data processing relies primarily on local devices rather than cloud servers; on the other hand, it usually does not need to establish a relationship with a remote cloud server, and it can meet the requirements of local users [6, 7]. The edge systems support fine-grained access to different dimensions of data [8]. In addition, mobile devices are faced with the problems such as the energy consumption of battery, the limited resources and the computing capacity in terms of the local processing [9]. Therefore, computation offloading has emerged, which can optimize the transmission delay of the task and reduce the user's computing burden [10]. However, if the users can offload all application tasks to MEC server, the server is likely to be overloaded. While studying the problem of computation offloading, the service cache is also an important topic for MEC [11]. The service cache prestores the database or library related to the application and allows the corresponding task to be offloaded. Due to the limited resources in edge servers, the caching decisions must be made carefully to maximize system performance [12]. However, how to use cache to reduce the service delay while maximize storage utilization is still a key issue in the edge network. But the heterogeneity of edge network and the

uneven distribution of users make it difficult for the system to balance cache and offloading. To deal with this problem, this paper proposes a cache-assisted offloading method and a target server matching strategy based on the cost of task to determine the appropriate target server to meet user's needs.

In brief, the main contributions of this paper are threefold:

(1) This paper considers the MEC server cooperative cache system. To reduce the access delay and the computation overhead, we design a new cache management strategy based on dynamic data approximate matching. Through an approximate matching algorithm based on the sample distance, a data set similar to the input data is selected from the collaborative cache space. By obtaining the corresponding calculation result for reuse, the cache hit ratio can be improved

(2) In order to improve the offloading efficiency, this paper proposes a new computation offloading method. According to the time sensitivity and communication cost, the optimal target server can be estimated; then the computation offloading problem can be transformed into an optimal path planning problem; finally, the optimal offloading path can be planned

(3) Evaluating the effectiveness of HCAM through specific simulation experiments

The rest of the paper is organized as follows. Section 2 summarizes the most related work. Section 3 introduces the system model in detail. In Section 4, we describe the cache-assisted offloading strategy. Section 5 introduces the efficiency evaluation. The simulation results are reported in Section 6. Finally, the conclusion and the future work are discussed in Section 7.

2. Related Works

In recent years, some augmented reality tasks have higher requirements for real-time performance when processing data. So, the traffic of mobile data continues to grow. It is not difficult to find that data requested by users is highly repetitive, which will lead to a large amount of redundant data transmission. In recent years, the caching problem has attracted the attention of researchers as a method to solve the delay problem [13, 14]. Cache is a new strategy to improve the performance and the service quality of mobile edge networks. It includes offloading tasks to the mobile edge cloud and storing computation results in the local storage located at the edge of network. This technology avoids redundant and repetitive processing of the same task, thereby simplifying the offloading process and improving the utilization of network resources [15, 16]. As a new method to alleviate the unprecedented network traffic, mobile edge caching has been widely used in the wired internet, and it has proved that it can reduce delay and energy consumption [17]. To date, a lot of research works have focused on optimizing caching methods to solve the delay and energy consumption problems in computation offloading. In [18], the author considered the horizontal cooperation between mobile edge nodes for joint caching and proposes a new transformation method to solve the problem of edge caching and improve cache hit rate of the network. In [19], authors designed a heterogeneous collaborative edge cache framework by jointly optimizing node selection and cache replacement in mobile networks. The joint optimization problem is expressed as a Markov Decision Process (MDP), and Deep Q Network (DQN) is used to solve the problem, which alleviates the offloading traffic load. In [20], the problem of edge cache optimization in fog radio access networks (F-RANs) was studied, and a distributed edge cache scheme was proposed, which reduced the delay of service and the traffic load. In [21], authors combined user's context behavior to optimize the cache and modeled the problem of maximizing the click-through rate of the content as a knapsack problem. In the MEC paradigm, a heuristic intelligent caching algorithm was proposed, which had the better cache hit rate and the stability and the lower overhead. In [22], authors studied the problem of vehicle edge caching in the actual vehicle scenes. In order to obtain the higher hit ratio, the service process was modeled as a joint process of vehicle movement and parking through the approximation theory, and a method based on the practical vehicle edge cache solution realizes the trade-off between hit ratio and interrupt request ratio. In [23], the computation offloading method of cached data was studied, and a new cache-aware computation offloading strategy was proposed. The goal was to minimize the equivalent weighted response time of all tasks with the constraint of computational power and caching capacity. In [24], the authors designed the underlying structure of cache causality and task's dependency model and designed an alternate minimization technique to reduce the complexity to alternately update the cache placement and the offloading decisions. In [25], the authors considered a complexed scenario, in which multiple moving MDs are sharing multiple heterogeneous MEC servers, and a problem named as minimum energy consumption problem in deadline-aware MEC system is formulated.

Some research works have concentrated on introducing the concept of edge caching in different systems, proposing the new frameworks or models to solve the optimization problem during offloading. In [26], a cooperative offloading and buffering model was designed, an optimization problem containing two independent problems was constructed, and a resource management algorithm was developed to guide a BS to jointly schedule the calculation of offloading and allocate the data buffers. The total delay of system communication can be minimized through the optimal offloading and caching decisions. In [27], authors proposed a collaborative edge caching scheme, defined the joint optimization problem as a Dual-Time-Scale Markov Decision Process (DTS-MDP), and proposed a framework based on Deep Deterministic Policy Gradient (DDPG). In [28], in view of the high link load of edge cache and the small storage space

of the server, a cloud-edge collaborative cache model based on the greedy algorithm was proposed. In [29], the problem of edge caching in the optical fiber computing networks was analyzed, and a capacity-aware edge caching framework was proposed. The problem of average download time minimization is described as a multiclass processor queuing process, and an algorithm based on the Alternating Direction Multiplier Method (ADMM) was proposed. In [30], a new intelligent edge is defined, which combines a heterogeneous IoT architecture with edge computing, caching, and communication. In [31], an offloading framework that enables task caching was proposed in edge computing to jointly optimize the response delay and the energy consumption of roadside units. In [32], in order to minimize the total delay consumption of tasks, the authors jointly considered computation offloading, content caching, and resource allocation as an integrated model, designed an asymmetric search tree, and improved the branch and bound method to obtain a set of accurate decision-making and resource allocation strategies. By summarizing the research of computation offloading method with cached data in MEC, we can conclude that the combination of edge caching and computation offloading has made progress in meeting user's requirements and improving user's experience.

In summary, most of the existing works do not take into account the influence of cache management and also not have full investigation of the collaboration of MEC servers. Thus, when the MEC environment changes dramatically, the burst request volume can bring sudden increased computation load to MEC servers, and the edge network links in certain regions will also become congested, leading to a significant impact on the efficiency of computation offloading. Accordingly, we take full use of the characteristics of edge cache to propose a computation offloading method based on cache-assisted, which can improve the cache hit ratio and the offloading efficiency.

3. Network Model

3.1. System Architecture. Computation offloading is a proven and successful example that can be used to enable resource-intensive applications on mobile devices. Efficient data sharing extends the collaboration capabilities of edge system [33]. For emerging mobile collaboration applications, when multiple users are at the same distance, offloaded tasks can be copied. Researchers urgently need to design a collaborative offloading scheme and cache popular calculation results that may be reused by other mobile users. In multi-acess mobile edge computing, tasks offloaded from the users are usually associated with the specific services, and these services need to be cached in MEC nodes to perform tasks. Deciding which services to cache and which tasks to perform on each MEC node with limited resources is critical to maximizing the efficiency of offloading [34]. In this section, considering an optimized regional collaborative cache system architecture. Table 1 presents the key notations of optimization model and corresponding descriptions.

As shown in Figure 1, considering a distributed multi-user MEC system consisting of multiple MEC servers connected via backhaul links, each of which can provide computation and storage power to meet the delay-sensitive requirements of tasks. This article assumes that only one task is generated per user. In this system, let $N = \{1, 2, \cdots, n\}$ denotes all users, and let $R_i = \{s_i, c_i, t_i\}$ denotes a random generating task i. s_i represents the size of tasks; c_i denotes the amount of computation resource needed to execute the application task, quantified in CPU cycles; t_i represents the time required to perform the task. In this system model, each BS is equipped with a MEC server to handle offloading requests. According to their own needs, the users can choose to perform tasks locally or offloaded to the edge servers. Assuming that each task occupies only one virtual machine, the user-generated request determines whether the virtual machine is occupied or not based on the offloading decisions. In this paper, the optimized cache management model and the offloading strategy are designed to reduce the user's request delay, which are introduced in the following sections.

Table 1: Symbol definition list.

Notions	Description
N	Number of users
M	Number of servers
s_i	Size of the task i
t_i	Processing time of task i
c_i	Computing resource required by task i
l_i	Computation offloading decision i
f_0	Computing capacity of the MEC server
T_i	The processing time of task i
T_n	The total processing time

3.2. Problem Formulation. Latency is an efficiency manifestation of system executing user's requests and a direct evaluation criterion of user experience. In this paper, the delay is composed of four parts: the communication time between MEC servers and users MT_{l_i}; the calculating time for servers to execute tasks CT_{l_i}; the waiting time for other tasks WT_{l_i}; and the time for BS forwarding to target MEC server BT_{l_i}.

Defining the offloading decision variables $L = \{l_1, l_2 \cdots, l_n\}$, a binary variable is used to represent the task executing locally or offloaded to the edge server:

$$l_i = \begin{cases} 1, & \text{the server caching resources,} \\ 0, & \text{other.} \end{cases} \tag{1}$$

Assuming that the channel adopts microwave link and the communication mode is full duplex, the calculation formula of the communication rate τ_i between the MEC server and the user is as follows:

$$\tau_i = w \log_2 \left(1 + \frac{P_i |h_i|}{I_i + n_0}\right), \tag{2}$$

where w is the channel bandwidth, P_i is the data rate sent by

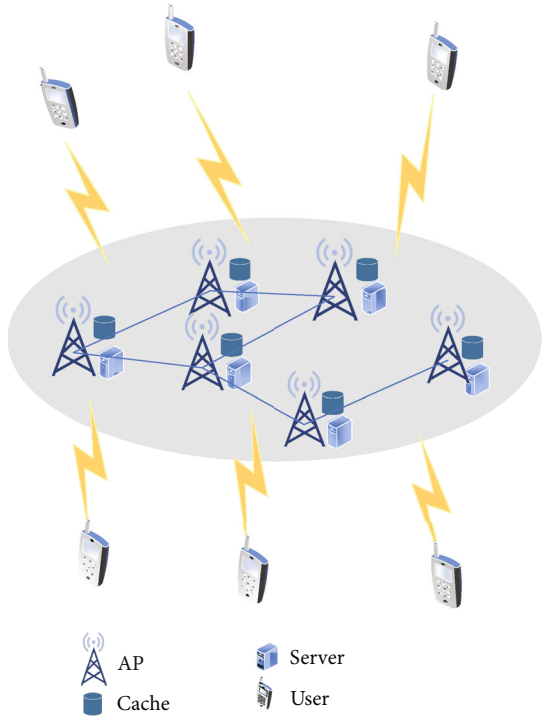

Figure 1: Cache-assisted MEC offloading architecture.

the user, h_i is the channel gain, I_i is the channel interference, and n_0 is Gaussian white noise. Therefore, the calculation formula of MT_{l_i} is:

$$MT_{l_i} = \frac{s_i}{\tau_i}. \tag{3}$$

The calculation formula of CT_{l_i}:

$$CT_{l_i} = \frac{c_i}{f_0}. \tag{4}$$

Because the MEC server accepts user requests on a first-come, first-served basis, the formula for WT_{l_i} is as follows:

$$WT_{l_i} = \sum_{i=1}^{k} \left(MT_{l_i} + CT_{l_i}\right). \tag{5}$$

k is the number of transmitted tasks before the task R_i. When $MT_{l_i} \geq t_i$, the server chooses to forward task i. The calculation formula of BT_{l_i} is as follows:

$$BT_{l_i} = \frac{s_i}{\varphi_i}. \tag{6}$$

φ_i is the forwarding transmission rate of BS. To sum up, the sum delay of task T_i is defined as the sum of MT_{l_i}, CT_{l_i}, WT_{l_i}, and BT_{l_i}. The calculation formula of T_i is as follows:

$$T_i = MT_{l_i} + CT_{l_i} + WT_{l_i} + BT_{l_i}. \tag{7}$$

The calculation formula of T_n is as follows:

$$T_n = \sum_{i=1}^{N} T_i. \tag{8}$$

This problem is equivalent to assigning task to the resource node in different region, minimizing the total processing time of all tasks. When the limited cache capacity of edge server is relaxed, it can be transformed into a classic transmission problem [35]. The optimization problem is as follows:

$$\min\left(T_n\right). \tag{9}$$

4. Optimal Offloading Solution

4.1. Content Update Method. In the MEC distributed architecture, a scenario of dynamic probabilistic cache is designed according to the time-varying content, and it can adapt to the time-varying content popularity without knowing the popularity. In the environment of the server collaboration, due to the different needs of users and the different requests cluster into the area of radius r, different regions are connected by optical fiber, and the collaboration area of MEC server can realize sharing of content. In the case of limited cache capacity, the MEC server must take the initiative to cache. The BS can parse part of the content request and place the cached content without returning the obtained result through the backhaul link, which relieves the pressure on the communication link. However, the popularity of the content changes with time, and the dynamic probability cache can adapt to the time-varying instantaneous content popularity and improve the cache hit rate of instantaneous content. In this article, the probability p_i of user is randomly requesting task i, $i \in (1, n)$ obeys Zipf's law, and therefore, p_i is calculated as follows:

$$p_i = \frac{1/i^{\eta}}{\sum_{i=1}^{n} 1/i^{\eta}}. \tag{10}$$

η is the value of the Zipf's distribution exponent.

4.2. Optimal Cache Management Strategy. In order to further improve the cache hit ratio, this paper adopts an optimized cache management strategy on the basis of the above content update method. Assuming that the area near each BS is divided according to empirical values, it is ensured that the number of edge servers in each area is approximately the same. Collaboration between servers can integrate cache at the edge of network. In a collaborative environment, the requested content can be transferred from one MEC server to another MEC server. As the computation capacity of edge servers are limited, repeated calculation of the same request will consume computing resources and increase the waiting

delay of end users [36]. The above process will face two challenges: on the one hand, since there are almost no two identical images and voices in the scene of image recognition and speech recognition, only the most similar data can be found instead of the completely identical data, so the traditional cache selection strategy based on the accurate matching is no longer applicable [37]; on the other hand, users generate a large amount of data every day, and it takes a lot of time to search for the same or similar data in the massive data, and the search becomes more difficult due to the increase of data dimensions. To address such problems, we propose a new cache management strategy based on the dynamic data approximate matching as given below.

Among these spatial index data structure construction methods, Baton-tree [38] is the most effective, and the complexity of other methods is affected by the dimension of data. When doing an approximate data look up with Baton-tree, it can get a similar data set of the input data, and then, the general approaches are to go through the similar data set and find the closest data set to input data and return the result but the search accuracy of that method is low. In order to improve the search accuracy, KNN [39] algorithm can be used to filter the data in the similar data set.

4.2.1. Matching Method Based on the Distance Threshold of Cache Data. The existing KNN search algorithm generally ignores the influence of distance on the accuracy of the algorithm and believes that approximate data has the same distance weight [40]. In fact, the distance between data in the set and the input data determines the similarity between the data and the input data. In this paper, a matching algorithm based on distance is proposed to search the data in the similar data set acquired by Baton-tree algorithm more accurately, so as to effectively improve the accuracy of data selection.

When defining the weight of each data, the matching method based on distance threshold takes into account the Euclidean distance between each similar data and the input data. Specifically, the farther the approximate data is from the Euclidean distance of the input data, the smaller the weight. Defining the Euclidean distance as $\mathrm{dis}(\mathrm{data}_0, \overline{\mathrm{data}_i})$, the formula is as follows:

$$\mathrm{dis}\left(\mathrm{data}_0, \overline{\mathrm{data}_i}\right) = \sqrt{(x_0 - \overline{x_i}) + (y_0 - \overline{y_i})}, \tag{11}$$

where data_0 denotes the input data R_0 and $\overline{\mathrm{data}_i}$ denotes approximate data of the input data R_0. Let (x_0, y_0) represents the coordinate of data_0, and let $(\overline{x_i}, \overline{y_i})$ represents the coordinate of $\overline{\mathrm{data}_i}$. Given the input data data_0 and the approximate data set $\mathrm{dataset}_j$, where $\forall \overline{\mathrm{data}_i} \in \mathrm{dataset}_j$, $\forall i \in j$. θ_i is used to indicate the weight value of the approximate data $\overline{\mathrm{data}_i}$, it can be calculated using the following:

$$\theta_i = \frac{j}{\left(\mathrm{data}_0, \overline{\mathrm{data}_i}\right)}. \tag{12}$$

θ_0 is the weight threshold.

In this paper, the discriminant function between data_0 and $\overline{\mathrm{data}_i}$ can be expressed as $\boldsymbol{p}_i$; it can be calculated using the following:

$$\boldsymbol{p}_i = \sum_{i=1}^{j} \theta_i = \sum_{i=1}^{j} \frac{j}{\mathrm{dis}\left(\mathrm{data}_0, \overline{\mathrm{data}_i}\right)}. \tag{13}$$

Let $|\boldsymbol{p}_i|$ denotes the coordinate axis vector modulo $\boldsymbol{p}_i$, vector $\boldsymbol{P}$ can be expressed as $\boldsymbol{P} = \sum_{i=1}^{j} \boldsymbol{p}_i$. Therefore, let λ_i defines the similarity between input data and data in the cache space; it can be indicated with cosine between $\boldsymbol{p}_i$ and $\boldsymbol{P}$; the formula is calculated as follows:

$$\lambda_i = \frac{\langle \boldsymbol{p}_i, \boldsymbol{P} \rangle}{|\boldsymbol{p}_i||\boldsymbol{P}|}. \tag{14}$$

λ_0 is the similarity threshold. For input data, data_i is similar to the data set obtained by the Baton-tree algorithm with different distances:

$$\mathrm{dataset}_j = \left\{(\mathrm{data}_1, \lambda_1, \theta_1), (\mathrm{data}_2, \lambda_2, \theta_2), \cdots, \left(\mathrm{data}_j, \lambda_j, \theta_j\right)\right\}. \tag{15}$$

The paper takes the maximum value $\lambda_{\max}$ among the j cosine values in $\mathrm{dataset}_j$, and the corresponding data value is denoted as $\mathrm{data}_{\max}$. If $\lambda_{\max} > \lambda_0$ and $\theta_i < \theta_0$, return θ_i and $\lambda_{\max}$ corresponding data_i as the approximate match of the input data data_0; otherwise, return Null, the query fails. As shown in Figure 2, it describes a cache management mechanism based on approximate matching.

4.3. Problem Transformation. According to the Dijkstra theoretical method [41], the problem of finding appropriate edge cache nodes can be transformed into the problem of shortest path planning. This paper assumes that the transmission rate between connected MEC servers, denoted as v , are all equal. Let $v_{m_2}^{m_1}$ be the transmission rate of the shortest route between the m_1 and m_2. If the server m_2 is connected with another sever m_3, there is a relationship among v, $v_{m_3}^{m_1}$, and $v_{m_2}^{m_1}$:

$$\frac{1}{v_{m_3}^{m_1}} = \frac{1}{v_{m_2}^{m_1}} + \frac{1}{v}. \tag{16}$$

It can be deduced from (16):

$$v_{m_3}^{m_1} = \frac{v_{m_2}^{m_1} \cdot v}{v_{m_2}^{m_1} + v}. \tag{17}$$

According to (16) and (17), it is obvious that the value of $v_{m_3}^{m_1}$ and $v_{m_2}^{m_1}$ proved that the shortest path means the least number of channels.

4.4. Offloading Location Confirmation. Some researchers have proposed different cost estimation methods of task execution. The most common methods are based on task time

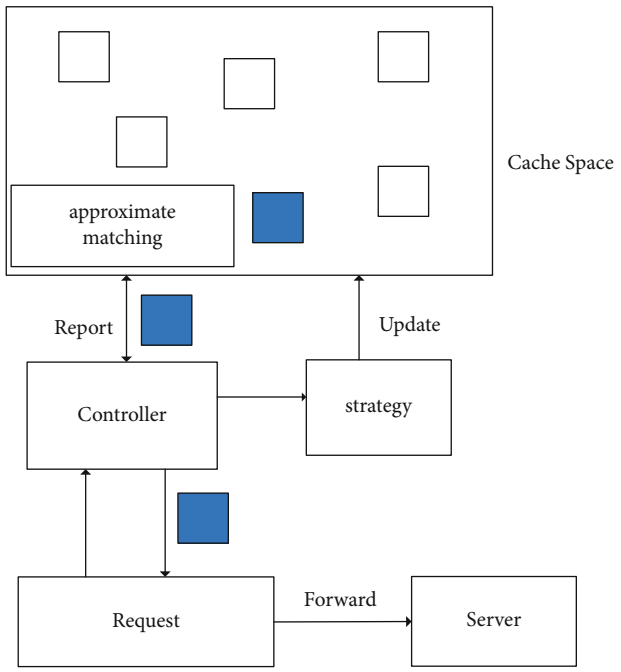

Figure 2: Cache-assisted MEC offloading mechanism.

sensitivity [42]. However, they did not consider the computing resource usage of the MEC server. Therefore, this paper proposes a new method for estimating the cost of task execution; the formula is as follows:

$$C_m = \frac{1}{Q}\sum_{i=1}^{m}(x_m - X). \tag{18}$$

It can obtain the congestion degree of the communication link of servers through (18), where Q is the number of divided areas, x_m is the number of nodes in the area m, and X is the average of the number of nodes in all areas. Combined with formula (8), the execution cost function of task is:

$$V_{i,m} = \alpha T_i + \beta C_m. \tag{19}$$

α and β are the weights given to the two objectives, respectively, with $\alpha + \beta = 1$. After the estimated execution time of the edge server and the congestion level of the communication link are returned to the terminal, the terminal device decides whether to perform a computation offloading and the computation offloading location. This article uses the following steps to confirm offloading position. It is described in Algorithm 1.

Avoiding resource conflicts: after the MEC server executes the search algorithm, it will find a server that meets the user's needs. Before assigning to users, because each user has different task requests and server processing time is also different, it will lead to some resource conflicts: when the server is idle, multiple users compete for cache and computing resources at the same time. This article considers that the server receives the user's resource request and sets the dynamic priority according to the execution time and order of the user request. Among them, the dynamic priority refers to obtaining the initial priority when the user applies for resources. The users constantly modify the priority level when using resources. In this way, conflicts in resource usage are avoided.

4.5. Offloading Path Identification. In respective areas, there are complex routes between edge cache nodes, and the least costly path needs to be found on the premise of determining the appropriate target server. This paper designs a path planning algorithm based on the cost of task. By using problem information to guide the search, the cost of the system search is reduced and the throughput is improved. It is shown in Algorithm 2.

5. Efficiency Evaluation

Figure 3 describes the performance comparison between the impoved distance search algorithm (IDSA) proposed in this paper and the distance search algorithm (DSA). By increasing the number of edge nodes, the total delay changes of two algorithms are compared. It can be seen that when the number of edge node is less than 4, the performance of two algorithms is close. Due to increase in the number of requested users, waiting, transmission, and calculation delay will all increase, resulting in the different degrees of increase in the delay of the two algorithms. However, it can be seen from the figure that when the number of servers is greater than 6, the performance gap between IDSA and DSA gradually increases, mainly because the algorithm proposed in this paper can quickly plan the offloading path and reduce the total delay of users. Therefore, the algorithm proposed in this paper is significantly better than DSA in terms of delay performance.

6. Simulation Experiment

In this section, we will evaluate the performance of the proposed scheme through simulation. The cache scheme is compared with the following four schemes: (1) random cache (RC) [43]: randomly caches popular content; (2) greedy cache (GC) [44]: only cache popular content in this area; (3) fair cache (FC) [45]: each collaboration area proportionally caches popular content; (4) collaborative edge cache offloading (CECO) [46]: only collaborative caching between edge servers; (5) heuristic cache-assisted method (HCAM): cache-assisted offloading method based on approximate matching proposed in this paper.

Experimental simulation parameters are shown in Table 2.

Figure 4 shows the relationship between the number of tasks and the total delay of under the same task processing method and different cache schemes. When the number of tasks is between 100 and 200, the system can process user requests in time, and the performance of each scheme is very close. When the number of tasks is greater than 200, the local and edge nodes cannot process all tasks within the time required by the user, and task access and waiting delays

Input: T_n; C_m; $A = \varnothing$
Output: Optimal offloading location E_m
for $i = 1$ to N **do**
  for $m = 1$ to $M - 1$ **do**
    Calculate the cost $V_{i,m}$ of the server m matching task i by formula (1)–(9);
    $V_{i,m} \longrightarrow A$;
  Select minimum $V_{i,m}$ corresponding m;
  return E_m

Algorithm 1: OLC Algorithm

Input: Request server E_o; cost function $G_1(m)$, $G_2(m)$; actual cost from starting point to candidate point $g_1(m)$, $g_2(m)$; estimated cost from candidate point to target point $h_1(m)$, $h_2(m)$; search step *step*
Output: Optimal path set p
$p = \varnothing$;
Obtain E_m through Algorithm 1, $E_o \longrightarrow p$, $E_m \longrightarrow p$;
If set p is empty **then**
  Return false.
Else
  While m is searched forward and backward and marked as Min and $g_1(\text{Min}) + g_2(\text{Min})$ is smallest **do**
  If(E_0, m) is null **then**
    Search for node $m + 1$;
  Else
    Calculate the cost of successor node m, $G_1(m) = g_1(m) + g_2(m)$;
    Select minimum $G_1(m)$, $E_0 \longleftarrow m$, $E_m \longrightarrow p$, Min $= m$, *step* $=$ *step* $+ 1$;
   If(Min, $m + 1$) is null **then**
    Search for the node $m + 2$;
   Else
    Calculate the cost of successor node $m + 1$, $G_1(m + 1) = g_1(m + 1) + g_2(m + 1)$;
    Select minimum $G_1(m + 1)$, Min $\longleftarrow m + 1$, $E_{m+1} \longrightarrow p$, Min $= m + 1$, *step* $=$ *step* $+ 1$;
   If(Min, $m + 2$) is null **then**
    Search for the node $m + 2$;
   Else
    Calculate the cost of successor node $m + 2$, $G_1(m + 2) = g_1(m + 2) + g_2(m + 2)$;
    Select minimum $G_1(m + 2)$, Min $\longleftarrow m + 2$, $E_{m+2} \longrightarrow p$, Min $= m + 2$, *step* $=$ *step* $+ 1$;
    Search backward from node E_s, search step is *step*;
    Return 7;
Return p;

Algorithm 2: OPP Algorithm.

increase, which in turn causes the total delay to increase with the increase of tasks. Compared with the greedy cache scheme, fair cache scheme, and random cache scheme, HCAM has a gap in the total delay of tasks as the number of tasks increases. When the number of tasks is 400, the gap is maximum. The task delay of HCAM scheme is 0.053 s, and the task delay of GC is 0.27 s. Although the performance of HCAM and CECO is relatively close, as the tasks increase, CECO has always been above HCAM. It can be seen from the figure that HCAM finally controls the task delay below 0.1 s; its performance is better than the other four schemes. This is mainly because the scheme adopts the principle of approximate matching to improve the cache hit rate when processing user requests at the edge and can reduce the transmission of the backhaul link, thereby reducing the user's waiting delay.

In Figure 5, when the number of user tasks is small, the four methods all show better optimization effect. When the number of tasks is about 100, because the user's request can be processed locally in time, it reduces task transmission and calculation time, so the local method performs better than offloading, CECO, and HCAM. It can be seen that when the number of tasks is between 100 and 200, the performance of HCAM and CECO is close to local, and three methods are better than the offloading. However, due to limited resources, as the number of user tasks increases, the total delay of the four schemes is increasing. When the number of tasks is greater than 200, the delay performance of four methods begins to show a gap. Finally, when the number of tasks reaches 500, HCAM is significantly better than the other three methods, and the performance gap is maximized. It can be seen that when there are many computing

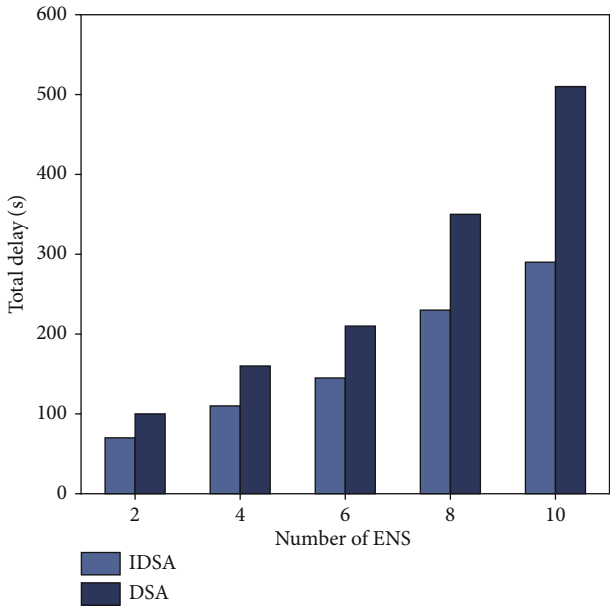

Figure 3: Delay comparison.

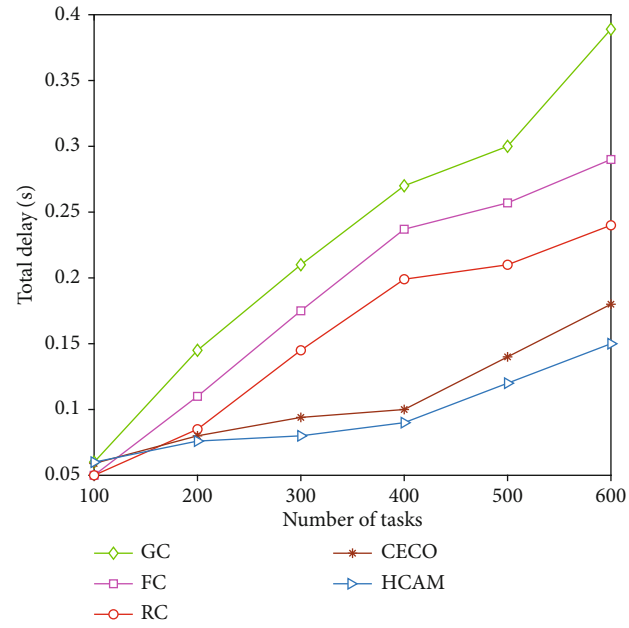

Figure 4: Delay comparison of different cache schemes.

Table 2: Simulation parameters.

Parameters	Value
Number of users	100
Number of edge nodes	10
Number of BS	1
Computing capacity of users	1-2 GHz
Computing capacity of MEC servers	6 GHz
The input data size of tasks	500-800 KB
Background noise power	-100 dBm
Channel bandwidth	1.5 MHZ
Cache size	150 GBs

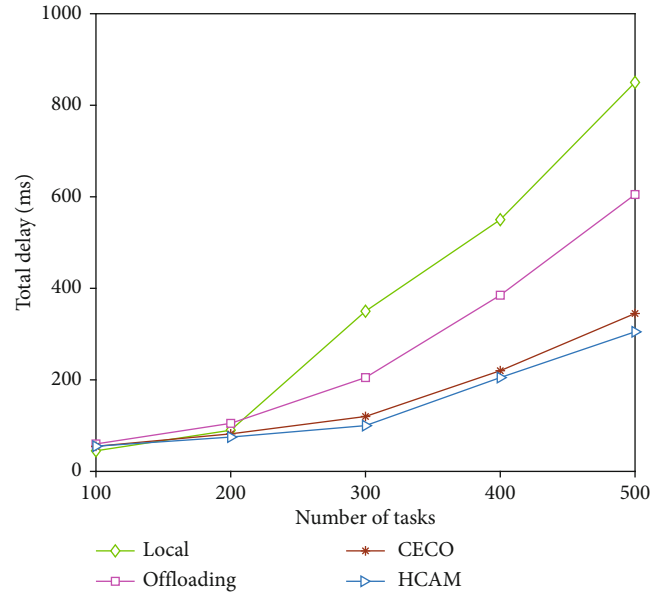

Figure 5: Delay comparison of different task processing schemes.

tasks, both HCAM and CECO use cache resources to reduce the transmission delay of the task; the performance is better than local and offloading. But when the cache is not hit, HCAM reduces the total delay by approximately 11.5% by forwarding tasks to the appropriate MEC server for processing in time which is compared with CECO.

Compared with GC, RC, FC, and CECO to verify the effectiveness of the proposed method, Figure 6 describes the comparison of the four methods on the cache hit ratio, and the cache hit rate is used as one of the performance criteria for evaluating the method proposed in this article. In the case of limited cache space, the higher the cache hit ratio, the lower the overall task processing delay. It can be seen from the figure that when the task time is relatively small, as the cache space increases, these four methods can all increase the cache hit ratio. As the number of tasks increases, the performance of the method proposed in this paper is better than other methods. The main reason is that HCAM optimizes the management and allocation of cache space through a new cache management mechanism. Compared with CECO, it adopts the cache approximate matching principle on the basis of edge collaborative cache, which improves the cache hit ratio.

Figure 7 shows the impact of cache size on the average system delay variation. Since five schemes adopt different cache management and allocation strategies, it can be seen from the figure that as the cache increases, the performance of the five methods differs in performance. As cache space increases, the cache of hot content will also increase, which increases the cache hit ratio. When the user generates a

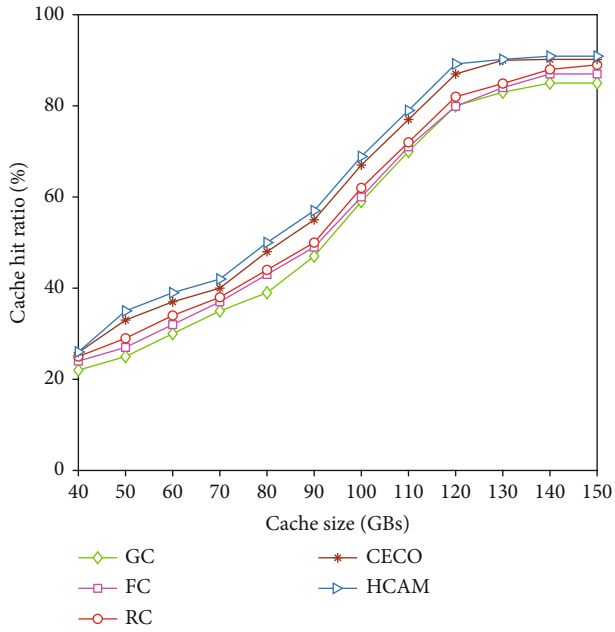

Figure 6: Comparison of cache hit ratio.

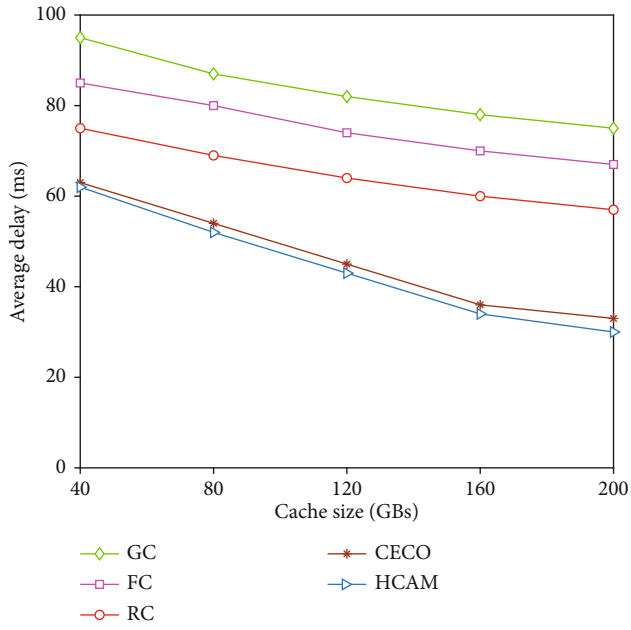

Figure 7: Comparison of average delay.

request, the edge server can directly send the cached content to the user. Users no longer need to wait for tasks to be offloaded to the target server, reducing transmission and calculation delays. When this article adopts an optimized cache management mechanism and cooperative cache model, even if the content requested by the user is not cached, it can be processed at the edge system as far as possible. Compared with the four methods, the average processing delay of the system is reduced to a certain extent. When the cache size is set to 40 GBs, the average delay of HCAM is the smallest. As the cache space increases, the GC method only satisfies the requests of a few users, and the system latency exhibits the greatest. Although FC has a higher cache hit ratio and better performance than GC, the average delay in the system is very close. The RC method further improves the cache hit ratio, which is better than the FC and GC methods. Although both HCAM and CECO use cooperative caching to reduce the average delay performance close to each other, HCAM uses the principle of approximate matching to increase the cache hit ratio, thereby reducing user access latency. However, when the cache space is 200 GBs, the performance of the HCAM method is optimal, which is about 1%, 24%, 36%, and 42% higher than the performance of CECO, RC, FC, and GC.

7. Conclusion and Future Work

In this paper, we focus on a computation offloading strategy. To reduce the processing delay, this paper design a new cache management strategy based on dynamic data approximate matching. Then, a new cache-assisted offloading mechanism for edge server is proposed. To improve the efficiency of offloading, this paper transforms the problem of offloading location confirmation into an optimal path planning problem, a heuristic algorithm based on task cost has been introduced to confirm the optimal server. The simulation results show that our scheme can reduce the total delay compared to GC, FC, RC, and CECO.

In the future, we will optimize the cache strategy. Besides, we will further study the computation offloading method under the job-related situation. In addition, we will explore algorithms suitable for task priority.

Data Availability

The (data type) data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China grants 61771289 and 61832012, Major Basic Research of Natural Science Foundation of Shandong Province with grants ZR2019ZD10; Key Research and Development Program of Shandong Province with grants 2019GGX101050.

References

[1] G. Zhang, S. Zhang, W. Zhang, Z. Shen, and L. Wang, "Joint service caching, computation offloading and resource allocation in Mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 99, pp. 1–1, 2021.

[2] L. Guangshun, S. Jianrong, W. Junhua, and W. Jiping, "Method of resource estimation based on QoS in edge

computing," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–9, 2018.

[3] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "IoT-based big data storage systems in cloud computing: perspectives and challenges," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75–87, 2017.

[4] Z. Cai and T. Shi, "Distributed query processing in the edge assisted IoT data monitoring system," *IEEE Internet of Things Journal*, vol. 99, pp. 1–1, 2020.

[5] A. Hekmati, P. Teymoori, T. D. Todd, D. Zhao, and G. Karakostas, "Optimal mobile computation offloading with hard deadline constraints," *IEEE Transactions on Mobile Computing*, vol. 99, pp. 1–1, 2019.

[6] T. Zhao, S. Zhou, L. Song, Z. Jiang, X. Guo, and Z. Niu, "Energy-optimal and delay-bounded computation offloading in mobile edge computing with heterogeneous clouds," *China Communications*, vol. 17, no. 5, pp. 191–210, 2020.

[7] S. K. Datta and C. B. Onnet, "An edge computing architecture integrating virtual IoT devices," *Consumer Electronics*, pp. 1–3, 2017.

[8] Z. Cai and X. Zheng, "A private and efficient mechanism for data uploading in smart cyber-physical systems," *IEEE Transactions on Network Science & Engineering*, pp. 1–1, 2018.

[9] P. Corcoran and S. K. Datta, "Mobile-edge computing and the internet of things for consumers: extending cloud computing and services to the edge of the network," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 73-74, 2016.

[10] F. Zhou and R. Q. Hu, "Computation efficiency maximization in wireless-powered mobile edge computing networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 5, pp. 3170–3184, 2020.

[11] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: new paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[12] H. Feng, S. Guo, L. Yang, and Y. Yang, "Collaborative data caching and computation offloading for multi-service mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 99, pp. 1–1, 2021.

[13] G. Li, J. Wang, J. Wu, and J. Song, "Data processing delay optimization in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2018, no. 1, pp. 1–9, 2018.

[14] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[15] Z. Cai and Z. He, "Trading private range counting over big IoT data," *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019.

[16] N. D. Pietro and E. C. Strinati, "Proactive computation caching policies for 5G-and-beyond mobile edge cloud networks," *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018.

[17] Y. Liu, D. Zheng, X. Xia, and B. Zhang, "Data caching optimization in the edge computing environment," *Environment*, 2020.

[18] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "A novel mobile edge network architecture with joint caching-delivering and horizontal cooperation," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 19–31, 2018.

[19] X. Wang, R. Li, C. Wang, X. Li, and V. C. M. Leung, "Attention-weighted federated deep reinforcement learning for device-to-device assisted heterogeneous collaborative edge caching," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 154–169, 2021.

[20] Y. Jiang, Y. Hu, M. Bennis, F. C. Zheng, and X. You, "A mean field game-based distributed edge caching in fog radio access networks," *IEEE Transactions on Communications*, vol. 68, no. 3, pp. 1567–1580, 2020.

[21] Y. Zeng, J. Xie, H. Jiang, G. Huang, and J. Li, "Smart caching based on user behavior for mobile edge computing," *Information Sciences*, vol. 503, pp. 444–468, 2019.

[22] Y. Zhang, C. Li, T. H. Luan, C. Yuen, and W. Wu, "Towards hit-interruption tradeoff in vehicular edge caching: algorithm and analysis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 99, pp. 1–13, 2021.

[23] H. Wei, H. Luo, Y. Sun, and M. S. Obaidat, "Cache-aware computation offloading in IoT systems," *IEEE Systems Journal*, vol. 14, no. 1, pp. 61–72, 2020.

[24] S. Bi, L. Huang, and Y. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 99, pp. 1–1, 2020.

[25] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[26] W. Fan, Y. Liu, B. Tang, F. Wu, and H. Zhang, "TerminalBooster: collaborative computation offloading and data caching via smart base stations," *IEEE Wireless Communications Letters*, vol. 5, no. 6, pp. 612–615, 2016.

[27] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 99, pp. 1–1, 2020.

[28] H. Tang, C. Li, Y. Zhang, and Y. Luo, "Optimal multilevel media stream caching in cloud-edge environment," *The Journal of Supercomputing*, vol. 10, pp. 1–20, 2021.

[29] Q. Li, Y. Zhang, Y. Li, Y. Xiao, and X. Ge, "Capacity-aware edge caching in fog computing networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 9244–9248, 2020.

[30] Y. Hao, Y. Miao, L. Hu, M. S. Hossain, G. Muhammad, and S. U. Amin, "Smart-edge-CoCaCo: AI-enabled smart edge with joint computation, caching, and communication in heterogeneous IoT," *Network, IEEE*, vol. 33, no. 2, pp. 58–64, 2019.

[31] C. Tang, C. Zhu, X. Wei, Q. Li, and J. J. P. C. Rodrigues, "Task caching in vehicular edge computing," *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021.

[32] J. Zhang, X. Hu, Z. Ning et al., "Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching," *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[33] X. Zheng and Z. Cai, "Privacy-preserved data sharing towards multiple parties in industrial IoTs," *IEEE Journal on Selected Areas in Communications*, vol. 99, pp. 1–1, 2020.

[34] J. Gao, S. Zhang, L. Zhao, and X. Shen, "The design of dynamic probabilistic caching with time-varying content popularity," *IEEE Transactions on Mobile Computing*, vol. 99, pp. 1–1, 2020.

[35] X. Meng, W. Wang, Y. Wang, V. Lau, and Z. Zhang, "Delay-optimal computation offloading for computation-constrained mobile edge networks," *2018 IEEE Global Communications Conference (GLOBECOM)*, 2019.

[36] N. Yousefian, J. Hansen, and P. C. Loizou, "A hybrid coherence model for noise reduction in reverberant environments," *IEEE Signal Processing Letters*, vol. 22, no. 3, pp. 279–282, 2015.

[37] S. L. Woodward, W. Zhang, B. G. Bathula et al., "Asymmetric optical connections for improved network efficiency," *J Opt Commun Netw*, vol. 5, no. 11, pp. 1195–1201, 2013.

[38] S. Surati, D. C. Jinwala, and S. Garg, "A survey of simulators for P2P overlay networks with a case study of the P2P tree overlay using an event-driven simulator," *Engineering Science and Technology, an International Journal*, vol. 20, no. 2, pp. 705–720, 2017.

[39] P. Dani and A. Thomas, "Bowditch's JSJ tree and the quasi-isometry classification of certain Coxeter groups," *Journal of Topology*, vol. 10, no. 4, pp. 1066–1106, 2017.

[40] Y. Li and B. Cheng, *An Improved K-Nearest Neighbor Algorithm and Its Application to High Resolution Remote Sensing Image Classification*, IEEE, 2009.

[41] O. A. Gbadamosi and D. R. Aremu, "Design of a modified Dijkstra's algorithm for finding alternate routes for shortest-path problems with huge costs," *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*, 2020.

[42] X. Q. Pham, T. D. Nguyen, V. D. Nguyen, and E. N. Huh, "Joint service caching and task offloading in multi-access edge computing: a QoE-based utility optimization approach," *IEEE Communications Letters*, vol. 99, pp. 1–1, 2020.

[43] W. B. Chu, L. F. Wang, Z. J. Jiang, and C. C. Chang, "Protecting user privacy in a multi-path information-centric network using multiple random-caches," *Journal of Computer Science and Technology*, vol. 32, no. 3, pp. 585–598, 2017.

[44] S. Ghandeharizadeh and S. Shayandeh, "Greedy cache management techniques for mobile devices," *IEEE International Conference on Data Engineering Workshop*, 2007.

[45] M. Kunjir, B. Fain, K. Munagala, and S. Babu, *ROBUS: Fair Cache Allocation for Multi-Tenant Data-Parallel Workloads*, Computer Science, 2015.

[46] Z. Qin, S. Leng, J. Zhou, and S. Mao, "Collaborative edge computing and caching in vehicular networks," *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, 2020.