Frend for Edge Servers: Reduce Server Number! Keeping Service Quality!

Pengmiao Li[1], Yuchao Zhang[1], Wendong Wang[1], Kaichuan Zhao[2], Bo Lian[2], Ke Xu[3], Zhili Zhang[4]
[1]*State key Laboratory Of Networking and Switching Technology*,
Beijing University of Posts and Telecommunications, Beijing, China
Email:pengmiaoli@bupt.edu.cn, yczhang@bupt.edu.cn, wdwang@bupt.edu.cn
[2]*Kuaishou Company, Beijing, China.* Email:zhaokaichuan03@kuaishou.com, lianbo@kuaishou.com
[3]*Computer Science and Technology, Tsinghua University, Beijing, China.* Email:xuke@tsinghua.edu.cn
[4] *Computer Science and Engineering, University of Minnesota, Minneapolis, US.* Email:zhzhang@cs.umn.edu

***Abstract*—To keep pace with the exploding data volume raised from geographical distribution edge networks, more and more edge servers have been built in recent years. As the computing power and storage capacity are different on each server, requests have to be transferred from one server to another before finally being responded and returned back to users. Such server-to-server transmission naturally introduce non negligible latency, which inevitably affects the quality of service (QoS). To eliminate this transmission latency, Internet Service Providers (ISPs) are building or renting more edge servers (both computing servers and storage servers) to reduce transmission distance and enhance the server configuration, which brings great costs. Fortunately, through a large number of real trace analysis, we found that it is possible to reduce server number while keeping the QoS!**

In this paper, we first disclose three key characteristics from Kuaishou Company: (1) Unbalanced request frequencies on different servers; (2) Imprecise latency measure on server-to-server transmission; (3) Nonlinear latency reduction to server number increment. Based on these findings, we propose a frequency-aware edge storage server deployment strategy *Frend* that is an improved Genetic Algorithm to optimize the number of edge storage servers by the internal diffusion capability that is a new latency measure called S2SL. Through a series of experiments using real application data, we demonstrate that while achieving the same S2SL, *Frend* can reduce the number of required edge storage servers by up to 56% compared with the state-of-the-art Anveshak method.

***Index Terms*—Edge Computing, Edge Storage Server, Edge computing Server, Server Deployment, Frequency Latency.**

I. Introduction

The previous decade has witnessed substantial growth in the Internet of Things (IoT) and mobile devices. With the explosion of quantity in the Internet of Things devices and mobile applications, the traffic scale of global data-center increases from 6.8 ZB to 14.1 ZB per year during 2016-2019. IoT devices will generate more than 175 ZB data by 2025 [1], which is an insupportable burden including limited storage, an amount of computation, and congested network for cloud data centers. Recently, edge computing technology plays an essential role in solving the issues mentioned above by edge servers deployed close to users. Data processed via edge servers is expected to account for 30% of the total data generated globally through creation, capture, and replication by 2025 [1].

As artificial intelligence evolves, computational tasks become increasingly time-consuming to finish [2]. These tasks are assigned to high computing power edge servers called edge computing servers (ECSs) to minimize the computation time. However, the ECS and the edge storage server (ESS) are generally not the same one because the limited storage of ECS can not cache all data that computing tasks need. This generates a transmission latency from Server-to-server that increases the completion time and inevitably affects the quality of service (QoS). Therefore, the problem to be solved in this paper is how to reduce the server-to-server latency, which is a kind of quick link problem between ECS and ESS, called the CSQ-link problem.

To solve this problem, Internet Service Providers (ISPs) can build or lease more edge servers to reduce transmission distances and increase server configurations, which results in huge costs and drives the need for a proper edge server deployment strategy. Though most of the existing researches is efficient in optimizing the latency of client-to-server and minimizing the cost of edge servers, they ignore the server-to-server transmission latency that inevitably affects the quality of service (QoS). Therefore, to ensure the interests of ISPs and the QoS, it is necessary to design a deployment strategy of ESSs to reduce cost while ensuring that the link between ECSs and ESSs is quick.

In this paper, we first disclose three key characteristics from Kuaishou Company [3]: (1) Unbalanced request frequencies on different servers; (2) Imprecise latency measure on server-to-server transmission; (3) Nonlinear latency reduction to server number increment. Based on these findings, we propose a frequency-aware ESSs deployment strategy called *Frend*, is an improved Genetic Algorithm to optimize the number of edge storage servers by the ESSs' internal diffusion capability that is a new latency measure called S2SL. *Frend* mainly includes three components: selecting candidates, choosing ESSs, and updating the status information of edge servers. Through a series of experiments using real applications trace, we demonstrate that while achieving the same S2SL, *Frend*

Yuchao Zhang (yczhang@bupt.edu.cn) is the corresponding author.

can reduce the number of edge storage servers required by up to 56% compared with the state-of-the-art Anveshak method. The contributions of this paper are summarized as follows:

- We investigate the problem of quick link between ECSs and ESSs, called CSQ-link problem.
- We introduce a new metric latency called S2SL to present the internal diffusion capacity of ESSs.
- Presenting *Frend*, a frequency-aware strategy to reduce the cost of ISPs deploys ESSs, is proposed.
- Demonstrating the benefits of *Frend* compare other algorithms by real traces.

In the next section, we introduce the background of the CSQ-link problem and the related work of server deployments with different optimization objectives. Section III illustrates perspectives about the potential performance that can be improved, including the number of ESS, the request frequency, and the metric latency S2SL. In section IV, it presents the formulation of the CSQ-link problem and the design of *Frend* architecture. Section V conducts experiments by detailed introduction and analysis of the results. And our conclusion is presented in Section VI.

II. Background and Related Work

A. Background

With the development of 5G and mobile Internet technology continuously, new applications appear constantly, such as AR/VR [4] and Internet of Vehicles [5], which is required more resources to ensure QoS for users. In addition to high bandwidth, these resources involve the rich computing and storage resources from edge servers are called ECS and ESS. To reduce the computation time accounted for a high percentage of the task response time, some edge servers with high computing powers are allocated to computation tasks usually. However, those edge servers cannot store all the data for each task because of their limited amount and storage volume, which makes the computing and storage resources usually supported by different ones, then generate to form ECSs and ESSs. Thus, it produces the transmission latency from ECSs to ESSs (server-to-server latency), which increases the task completion time and affects the user's QoS.

To eliminate this transmission latency, ECSs need to obtain data required from ESSs quickly when completing the computing tasks, which is the CSQ-link problem proposed in this paper. It is easy to assume that the more ESSs deployed in an application, the faster the ECS can get computational data required, but the cost to the ISP is enormous. Therefore, suitable edge servers need to be selected as ESSs for these ECSs to reduce server-to-server latency, i.e., deploying (renting) ESSs for ISPs in many edge servers to ensure low cost and high performance.

B. Related Work

In studies about the deployment of edge servers, researchers have proposed many solutions from different perspectives: (1) minimizing the access latency and balancing the workload of edge servers [6]–[18]; (2) minimizing the cost of edge servers while limiting maximal latency [19]–[25], which include maintenance cost and the number of the request edge servers; (3) and others [26]–[29], such as the computing capacity. Presenting some related works in two-part: Latency & Workload, and Cost & Number, which are the main factors of this paper concern.

1) Latency & Workload: The latency between the edge servers and users & the workload of edge servers are main KPIs and attracted the attention of some researchers in edge server deployment. Manasvi [6] proposes a Social network Aware Dynamic Edge Server deployment (SADES) stregy with low latency communication, which uses the information from the overlay social network groups to efficiently identify a few influential base stations to place the edge servers. Paper [7] designs a novel algorithm, called PACK, to reduce application latency and data transfer load in opportunistic Internet of Things systems. Wang [8] minimizes the access latency and balance the workloads of edge servers by adopting mixed integer programming to study the edge server deployment problem in mobile edge computing environments for smart-cities. Yan [9] proposes an approximate approach that adopt the K-means and mixed-integer quadratic programming to balance the workload between edge clouds and minimize the service communication latency of mobile users.

2) Cost & Number: The cost-efficient of deploying edge servers becomes a crucial research field, which gains many researchers' focus. Yin [19] presents Tentacle, a decision support framework to provision edge servers that can discover proper unforeseen edge locations by considering various pragmatic concerns in edge provisioning, which significantly improves the efficiency and reduces the cost of edge provisioning. The number of edge servers deployed affects the application service providers' cost. M [21] chooses an appropriate number of nodes for deployment based on a fixed cost. Paper [22] considers the maximum capacity of edge servers and provides a partition of the edge node cluster, which integrates as much communication as possible at the edge, is determined by maximizing the number of connections with one area. Some papers choose the lowest number of servers [23] [24] by setting the highest client-to-server latency tolerated, which can increase the workload of edge servers, improve resource utilization, and reduce the number of edge servers.

Although these solutions can effectively optimize the latency of client-to-server and the cost of edge servers, they ignore the server-to-server latency that leads to high completion time and affects the user's QoS. Therefore, to ensure the interests of ISPs, it is necessary to design a deployment strategy of ESS to reduce costs (the number of ESSs) while ensuring that the link between ECSs and ESSs is quick.

III. Potential of Deployment ESSs

To find a potential ESSs deployment strategy that can reduce cost and the server-to-server latency, we analyze the real trace from Kuaishou and find three key characteristics: (1) Unbalanced request frequencies on different servers, (2) Im-

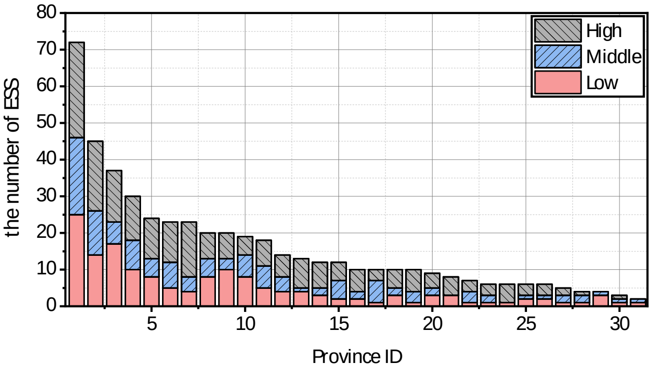

Figure 1. ESSs access frequency information in 31 regions.

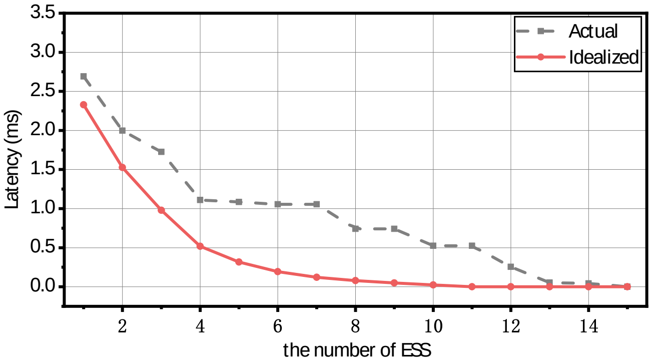

Figure 2. Traditional Latency and S2SL.

precise latency measure on server-to-server transmission, and (3) Nonlinear latency reduction to server number increment.

A. *Unbalanced request frequencies on different servers.*

Theoretically, a region with high request frequency usually needs to deploy more ESSs for applications to ensure the user's response time. However, some applications have many less frequently requested ESSs in each region, which leads to waste storage resources and thus higher costs for ISPs. Figure 1 shows that the number and the request frequency of each ESS in Kuaishou from 31 provinces of China, where a bar is composed of three sub-bars with different colors (red, blue, and black). The bar represents the total number of ESSs in one province, while the sub-bars represent the ESSs number with different request frequency range *Low*, *Middle*, and *High*. In this paper, we treat request frequency under 1,000 as *Low*, *Middle* is from 1,000 to 100,000, and others as *High*. For example, the total number of ESSs deployed is 72, where the number of ESSs in *Low*, *Middle*, and *High* are 25 (red sub-bar), 46-25 = 11 (blue sub-bar), and 72-46 = 26 (black sub-bar) respectively. *Middle* is 46-25 = 11 (blue sub-bar), and *High* is 72-46 = 26 (black sub-bar). It is not difficult to find that the request frequency of many ESSs is less than 1,000 (red sub-bar) in each province (such as province 1), which leads to low utilization in low-frequency ESSs. Therefore, these ESSs with low frequency have higher promotion potential than others by reducing the ESSs number via removing low-frequency ESSs, which can improve the workload (frequency requested).

B. *Imprecise latency measure on server-to-server transmission.*

According to the analysis, we find the greater internal diffusion capacity of ESS, which is the new metric latency based on the request frequency and called S2SL, the lower server-to-server transmission latency of ESS. The traditional theoretical latency metric computes the transmission latency by requesting ESSs once, which ignores the total latency from more times request. For example, there are 3 ESSs: A, B, and C with request frequency 10, 50, and 20, respectively, and the latency of A-B is 3, B-C is 5, and A-C is 4. The traditional theoretical latency is (3+5+4)/3=4 (Actual). The based frequency S2SL is equal (3*10+5*50+4*20)/(10+50+20)=4.5, so the traditional latency is unequal S2SL. In Figure 2, we calculate the traditional theoretical latency and the lowest transmission latency S2SL that from the optimal solution to deploy ESSs under a different number. It is easy to find that the Idealized latency (red line) as S2SL is superior to the Actual latency (black line). Based on the traditional theoretical measure to deploy ESSs for applications, the actual latency will be higher than the idealized. Therefore, using S2SL to select ESS is a potential opportunity to reduce the transmission delay of data required to obtain ECS tasks from ESS, thus reducing the task's completion time and improving the user's response speed.

C. *Nonlinear latency reduction to server number increment.*

In theory, applications could rent huge ESSs to reduce the transmission latency of data acquired by ECS. However, this is not a wise move, as analysis of the Kuaishou trace shows that latency does not decrease linearly with the number of ESSs increasing, and the result shown in 2 (red line). The transmission latency of server-to-server is high when choosing the number of ESSs in a range (as 1 to 4), and its rate of decline gradually tends to flatten out in someone range such as the number 5. This means that even if we choose to lease a large number of servers, not only is the performance increase minimal, but the cost is enormous. Therefore, there is a potential perspective for reducing the cost of ISPs by deploying a fit number of ESSs under add to a tiny latency with user acceptable that we called it is without affecting QoS.

Based on the above analysis, reducing the cost of ISPs can adopt to three potential perspectives including *(1) Balancing the request frequency of ESSs*, *(2) S2SL as a metric latency to deploying ESSs*, and *(3) Choosing the suitable number of ESSs without effecting QoS*. Designing a deployment strategy for edge storage servers through these perspectives is the focus of this paper and is described in detail for Section IV.

IV. System Models and Problem Formulation

In this section, we first introduce and formulate the quick link problem between ECSs and ESSs (subsection IV-A), and

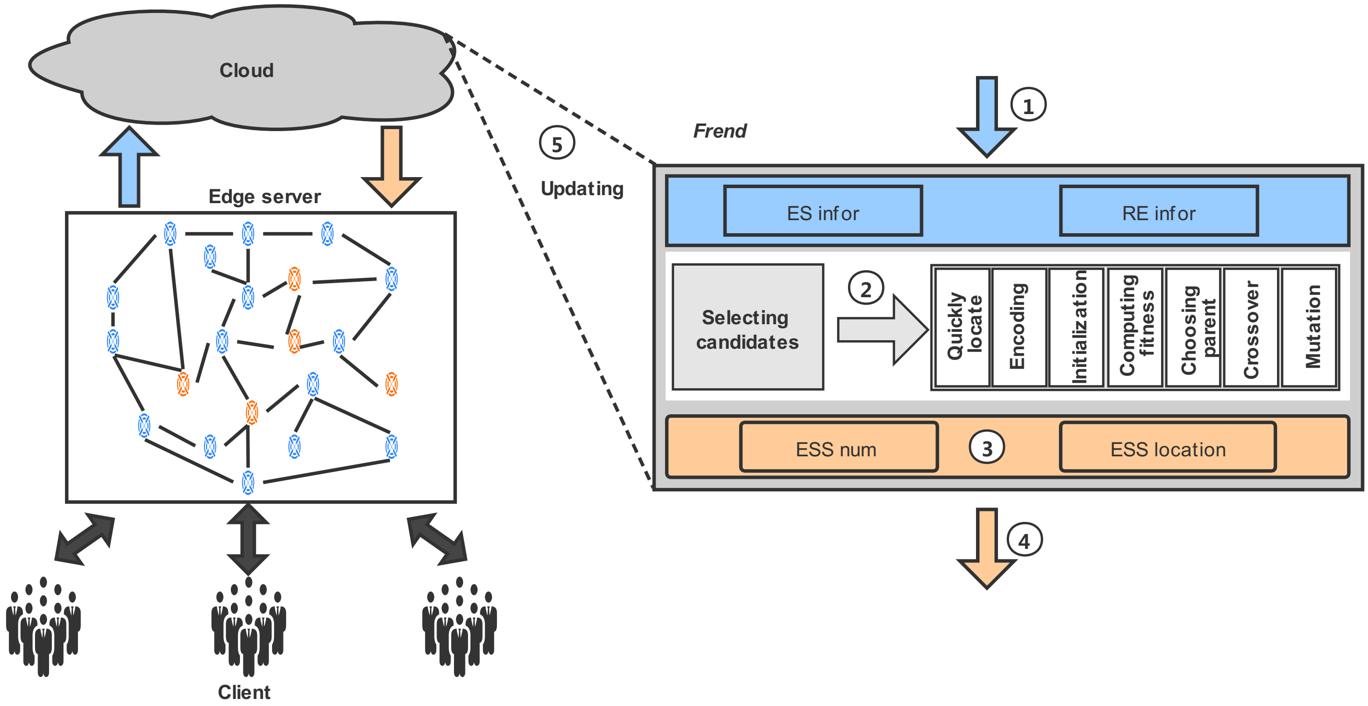

Figure 3. The framework overview.

then propose the deployment strategy *Frend* to minimize the number of ESSs and reduce S2SL in subsection IV-B. Finally, we give an overview of the framework in subsection IV-C.

A. *Problem Formulation*

The CSQ-link problem described in subsection II-A is how to select some edge servers as ESSs for the application among all edge servers, which minimizes the ESSs number required under the S2SL constraints ζ. Edge server set are denoted by $X = \{x_1, x_2, ..., x_N\}$, and $S = \{s_1, s_2, ..., s_m\}$ is defined by the ESSs set. Let $S2SL = \{s2sl_1, s2sl_2, ..., s2sl_N\}$ denotes each edge server's S2SL computed by equation (1).

$$s2sl_i = \frac{\sum_{j=1}^{N} l_{i,j} \times f_i}{\sum_{j=1}^{N} f_i}. \tag{1}$$

N is present the total number of edge servers, and f_i is the frequency requested of edge server i from clients. $l_{i,j}$ is the latency between edge server i with edge server j is computed by

$$l_{i,j} = min(dis(i, j), dis(i, p) + dis(p, j)), \tag{2}$$

where $\forall i, p, j \in S$. According to the statements above, the CSQ-link problem can be formulated as follows:

$$\min \quad \sum_{i=1}^{N} x_i \tag{3}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} x_i \times s2sl_i \leq \zeta, \quad \zeta > 0 \tag{4}$$

$$x_i \in \{0, 1\}, \qquad \forall i \in N \tag{5}$$

Although it is possible to solve this problem and find the optimal solution by computing all solutions, it is very time-consuming because the solution space makes the characteristic of exploding up with the enlargement of the number of edge servers. Therefore, we propose an improved Genetic Algorithm that can find the optimal solution to solve the CSQ-link problem with high efficiency in the current environment, which is called *Frend* and is introduced detailedly in the next subsection.

B. *Frend Strategy*

The core of *Frend* is an ESSs deployment strategy that selects a solution from all solutions that minimizes the number of ESSs under the user-tolerated S2SL. *Frend* is mainly composed of three components *Selecting candidates*, *Choosing ESSs*, and *Updating the state of edge servers* as follows.

*1) **Selecting candidates**:* To reduce the *Frend*'s computing time, some edge servers selected as the ESSs' candidate set $\hat{S}$ is necessary, which are high possibility to become ESSs than other ones. Selecting a candidate set of ESSs into two parts in this paper. On the one hand, the edge servers are divided into several regions $R = \{r_1, r_2, ..., r_o\}$ based on the location of the edge servers and clients. Some edge servers with high latency are removed because these are too far away from clients, and the remaining edge servers recognize as the initial candidate set of ESSs for region o. On another hand, many edge servers among the initial candidate set have been occupied by other ISPs, which can be filtered out those based on global information (state, location, and request frequency et.al) $G = \{g_1, g_2, ..., g_N\}$ from the cloud. The remaining edge servers $\hat{X} = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_N\}, \forall r \in N$ as the ESSs' candidate set are obtained by equation (6).

$$\hat{x}_i = \begin{cases} 1, & \text{if } g_i \geq \hat{g} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$\hat{g}$ denotes the threshold of allowable available resources of edge server. (Line 1-2 in algorithm 1)

*2) **Choosing ESSs:*** Genetic Algorithm not only can jump out of the local optimal solution but also can have low computation time compared with other heuristic algorithms. However, it also has some shortcomings, such as depending on the original population and the parallelism potential of the algorithm is not stimulated which are impact the computation time. Therefore, to improve *Frend*'s performance, we respectively design two methods include *Quickly locating* and *Reducing the repeat computation workload* for addressing above shortcomings.

- **Quickly locate**.
  Narrowing the candidate set dictionary of ESSs by selecting candidates component, but it is difficult and time-consuming to compute the best ESSs set for ISPs. Because it includes deciding the edge server number as ESS and selecting which one is ESSs set from the 2^N combinations (chromosomes) of edge servers. The candidate number of edge servers in the ESS set is $E = \{1, 2, ..., N\}$. Let $U = \{u_1, u_2, ..., u_v\}$ denotes the candidate set for the ESS number of their solutions. To reduce the computation time, we lower the number of E dictionary by pre-computation quickly, which is called *Quickly locate*. *Quickly locate* adopts to a Binary Search algorithm to find two edge server combinations (chromosomes) are s'_i and s'_j with S2SL $s2sl_i$ and $s2sl_j$ respectively, where $s2sl_i < \zeta < s2sl_j$ and $\left|num(s'_i) - num(s'_j)\right| = v, \forall v \in \mathbb{N}^+$. It gives powerful support to reduce the computation to find the best ESSs set from huge combinations in the Genetic Algorithm. That decreases the dependence on the original population and then reduces the computation time.
- **Reducing the repeat computation workload.**
  Although the computation time is reduced by quickly locating, there is a lot of repeat computation workload in choosing the best ESSs set. Such as, if u_i and u_j of s'_i and s'_j are 5 and 6 receptively, the s'_j's S2SL can be compute by adding one edge server based on s'_i , which means double-counted in s'_i. Therefore, to *reduce the repeat computation workload*, we calculate the $s2sl_i$ of $u_i = 5$ as well as save the $s2sl_i$ as a part of $s2sl_j$ of $u_i = 6$. That means this method can achieve partial parallelism and then reduce the total computation time of finding the best ESSs set for ISPs.

By two steps above, we gain ESSs candidates' number of the solution and reduce the repeat computation workload. Finally, we combine 5 levels (Figure 3) including encoding, initialization, computing fitness, choosing parents, crossover, and mutation to find the optimal ESS set. Gene and chromosome fitness are defined by each edge server and S2SL. The chromosome development tends to a lower S2SL through choosing parent, crossover, and mutation. The detailed steps for obtaining the optimal solution are shown in line 3-22 of algorithm 1.

*3) **Updating the state of edge servers:*** To ensure the information of global edge servers is consistent that can prevent the ones chosen as ESSs repetitively for different ISPs, we need to update the state information of edge servers in the cloud periodically (Line 23 in algorithm 1). Updating the ESS set periodicity determined by client dynamic access patterns [30] is necessary to improve users' Qos, which is the focus of our future work. At last, we summarize *Frend* in Algorithm 1.

Although we can obtain the solution to choose edge servers as ESSs for ISPs through *Frend*, it is significant that how to layout and apply *Frend* in the edge, which can affect the performance of ESSs for ISPs. Therefore, we introduce the Framework Overview of *Frend* in the next subsection.

C. Framework Overview

The edge networks that complete the edge computing task mainly include three parts: Client, Edge Servers, and Cloud, as shown in the left of Figure 3. The edge servers serve a client when it is requested for the computing task. If the edge server can meet the task requirements, it will directly return the task result to the client. If the task requires not to meet, it will go to the cloud for task calculation and then return the calculation result. Our algorithm *Frend* is deployed in the cloud to obtain the server's information that is calculated based on the client frequency requested and the S2SL.

The workflow of the whole architecture is as follows:

- Obtaining the frequency accessed of each edge server through the access history data (①);
- Choosing ESSs by *Frend* (②);
- Calculating the edge server number of ESS set and the ESSs' location (③);
- Deploying the content according to the edge servers chosen as ESSs (④);
- Updating the ESSs periodically (⑤).

V. Evaluation

In this section, we evaluate our approach *Frend* by real traces from Kuaishou company, show the results of applying *Frend* on them versus the existing representative policies, and finally analyze the results in the average frequency of provinces and the frequency requested range of ESSs.

Algorithm 1: The pseudo code of *Frend*

```
Input: The limit of the S2SL $\zeta$;
    The latency $l_{i,j}$ between edge server $i$ with $j$;
    The total edge servers $X$ including $N$ edge servers;
    The frequency requested of edge servers is $F$;
    The state of edge servers is $G$.
Output: the lowest number $u'$ and set $s'$ of ESSs.
$\hat{S} = Preliminary(G, X)$ ;
Calculating $\hat{X}$ by equation (6);
$U = BinarySearch(F, l_{i,j}, \hat{S})$;
Initializing the population with $u' = u_1$ edge servers;
while True do
    evaluate the fitness $s2sl'$ of all chromosomes in population;
    if $min(s2sl') <= \zeta$ then
        $s'$ = the chromosomes with the lowest $s2sl'$;
        if $min(U) == u'$ then
            while True do
                pop chromosome from matingPool and elitePool $u' = u' - 1$ ;
                Cross over and mutation generate the population with $u'$ edge servers ;
                evaluate the fitness $s2sl'$ of all chromosomes in population;
                if $min(s2sl') >= \zeta$ then
                    $u' = u' + 1$ ;
                    Return $u'$,$s'$
                updating $s'$ ;
        else
            Return $u'$,$s'$
    else
        append chromosome (edge server) to matingPool and elitePool $u' = u' + 1$ ;
        Cross over and mutation generate the population with $u'$ edge servers;
update G periodically.
```

A. *Experiment setting.*

Algorithms:

In this experiment, we compare *Frend* with three algorithms that are based on the traditional theoretical latency metric introduced in subsection III-B, which are *Random*, *GA*, and *Anveshak* as follows.

- *Random*: The deployment strategy *Random* [31] chooses servers to fit the request number of ISPs randomly, and then we choose the random servers as ESSs.
- *GA*: *GA* [32] based on latency between edge servers to choose the lowest one to deploy, and then we regard the lowest one above as ESSs.
- *Anveshak*: [23] based on latency between edge servers to choose one that has single hops of the smallest latency to provide the service for ISPs, and then we choose those edge servers as ESSs.

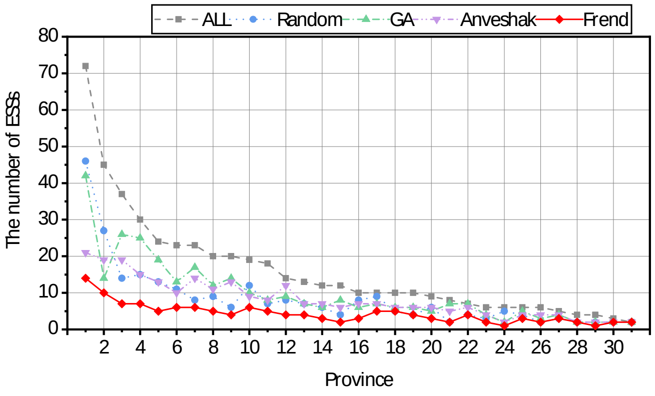

Figure 4. [Performance comparison] Comparison of the number of chose ESSs under the same S2SL.

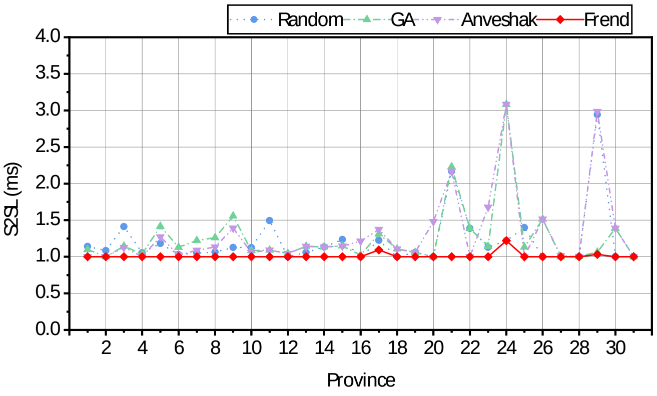

Figure 5. [Performance comparison] Comparison of S2SL under the same number ESSs.

Datasets: Datasets includes two parts as follows.

- *Access frequency:* The experiment dataset is one hour's real trace from Kuaishou [3], which includes 488 edge servers are deployed at 31 provinces in China. The total number of requested items exceeds 12 million, and the number of short videos exceeds 10 million [33].
- *Edge server latency:* We randomly generate the network topology and the latency between all the edge servers based on the different provinces above as the edge server latency information.

B. *Performance comparison.*

Comparing the ESS number with the same S2SL. Figure 4 shows the ESS number chosen by different algorithms under the same S2SL, in which the abscissa represents the provinces ID, and the ordinate represents the request number of ESS. The black line is the total number of ESSs from original data (kuaishou Datasets), while blue, green, purple, and red represent the ESS number are chosen by *Random*, *GA*, *Anveshak*, and *Frend* algorithms, respectively. It is not difficult to find that the number of ESSs in our method is

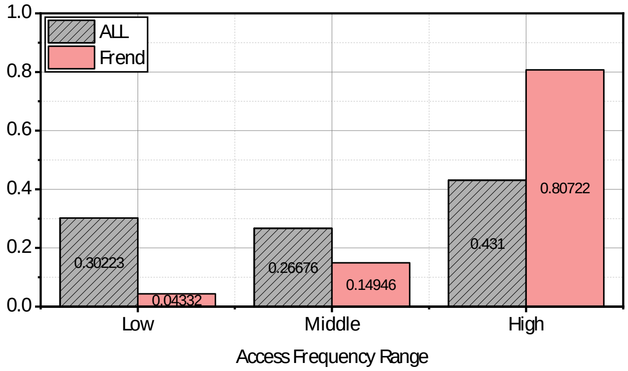

Figure 6. [Result Analysis] Access frequency range of all ESSs.

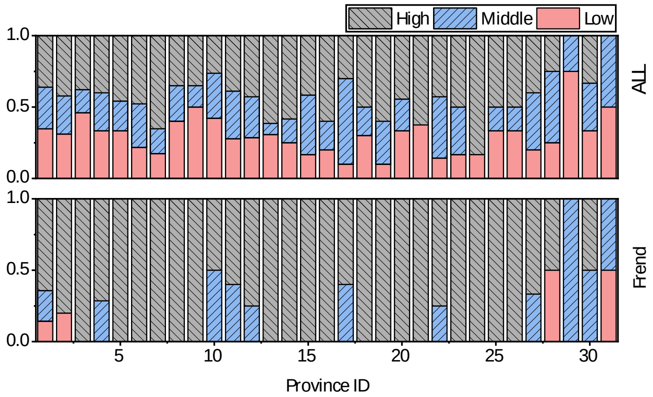

Figure 7. [Result Analysis] The access frequency distribution of ESSs in 31 provinces.

reduced by up to 80%, 56% compared to the original data and *Anveshak*. Our method is superior to other baselines in each province with the same S2SL.

Comparing the S2SL under the same ESS number. To show the flexibility of our strategies *Frend*, we compare the S2SL of different algorithms under the same number of ESSs and the result is shown in Figure 5. The abscissa represents the province's ID and the ordinate represents the ESSs set S2SL. Blue, green, purple, and red lines respectively represent the S2SL computed by *Random*, *GA*, *Anveshak*, and *Frend* algorithms. It is not difficult to find that our method *Frend* has the lowest S2SL compared with other algorithms in each province with the same number of ESSs to provide service for ISPs. For example, the S2SL is about 1.2 ms from *Frend* and is about 3 ms from *Anveshak* in province 29, that the former algorithm has 60% less latency than the latter one.

C. *Result Analysis.*

In Chapter III-A, we find that about 30% of ESSs have an access frequency of less than 1,000. These ESSs are extremely low resource utilization, which brings the chance to balance the request frequency and reduce the number of ESSs with low frequency requested by *Frend*.

Figure 6 shows a comparison of the range of ESSs' frequency requested from *Frend* result (red bar) and real trace (black bar). We can find from this figure that the *Low* percentage of ESSs drops from 0.30 to 0.04, which means the number of ones removed is more than 86.7% from ALL. Middle dropped from 26% to 15%, and *High* increased from 43% to 80%. The remaining ESSs with low frequency requested cannot be removed because the total number of requested in this region is low.

To clearly present the ESSs chose information of the frequency requested range, we show the average frequency distribution of ESSs in Figure 7. This figure is divided into two parts: bottom and top, which present the access frequency in 31 provinces from ALL ESSs and *Frend* chose one, respectively. From the point of view of each province, the proportion of low frequency ESSs (red bar) is close to zero at the bottom and the proportion of high frequency ESSs (black bar) at the bottom is higher than the top one. We can hardly see the red bar at the bottom of Figure 7.

The ESSs deployment strategy based on the edge server frequency requested information, *Frend*, is verified by experiments under real application request frequency. *Frend* can choose the ESS with low S2SL among many edge servers. That means we can reduce the number of ESSs and optimize the server-to-server latency by proposed *Frend*. Therefore, these ESSs with low S2SL can keep the QoS.

VI. Conclusion

In this paper, we investigate the CSQ-link problem and introduce a new metric latency called S2SL. To resolve the problem above, we analyze of real trace from Kuaishou and find three characters: (1) Unbalanced request frequencies on different servers, (2) Imprecise latency measure on server-to-server transmission, and (3) Nonlinear latency reduction to server number increment. Based on the characters above, we propose a frequency-aware strategy *Frend* to minimize the request number (the cost of ISPs) of ESSs for a single application while ensuring low S2SL. Finally, we evaluate the *Frend* strategy using real application data and demonstrate that while achieving the same latency, *Frend* can reduce the number of required edge storage servers by up to 56% compared with the state-of-the-art Anveshak method. Therefore, we can reduce the costs of ISPs by *Frend* to deployment ESSs with low S2SL that can keep the QoS.

Acknowledgment

The work was supported in part by the National Natural Science Foundation of China (NSFC) Youth Science Foundation under Grant 61802024, BUPT-Chuangcache Joint Laboratory under B2020009, the Key Project of Beijing Natural Science Foundation under M21030, the National Key R&D Program of China under Grant 2019YFB1802603, AND the NSFC under Grant 62072047. The work of Pengmiao Li was supported in part by the BUPT Excellent Ph.D. Students Foundation under CX2019134.

References

[1] R. Davi, G. John, and R. John, "The digitization of the world from edge to core," https://www.idc.com/[R/OL], 2018.

[2] R. Singh, A. Dunna, and P. Gill, "Characterizing the deployment and performance of multi-cdns," in *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*. ACM, 2018, pp. 168–174. [Online]. Available: https://dl.acm.org/citation.cfm?id=3278548

[3] kuaishou, "Kuaishou," https://www.kuaishou.com, 2019.

[4] S. N. B. Gunkel, R. Hindriks, K. M. E. Assal, H. M. Stokking, S. Dijkstra-Soudarissanane, F. B. ter Haar, and O. Niamut, "Vrcomm: an end-to-end web system for real-time photorealistic social VR communication," in *MMSys '21: 12th ACM Multimedia Systems Conference, Istanbul, Turkey, 28 September 2021 - 1 October 2021*. ACM, 2021, pp. 65–79. [Online]. Available: https://doi.org/10.1145/3458305.3459595

[5] X. Shen, R. Fantacci, and S. Chen, "Internet of vehicles," *Proc. IEEE*, vol. 108, no. 2, pp. 242–245, 2020. [Online]. Available: https://doi.org/10.1109/JPROC.2020.2964107

[6] G. Manasvi, A. Chakraborty, and B. S. Manoj, "Social network aware dynamic edge server placement for next-generation cellular networks," in *2020 International Conference on COMmunication Systems & NETworkS, COMSNETS 2020, Bengaluru, India, January 7-11, 2020*. IEEE, 2020, pp. 499–502. [Online]. Available: https://doi.org/10.1109/COMSNETS48256.2020.9027421

[7] T. Lähderanta, T. Leppänen, L. Ruha, L. Lovén, E. Harjula, M. Ylianttila, J. Riekki, and M. J. Sillanpää, "Edge computing server placement with capacitated location allocation," *J. Parallel Distributed Comput.*, vol. 153, pp. 130–149, 2021. [Online]. Available: https://doi.org/10.1016/j.jpdc.2021.03.007

[8] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distributed Comput.*, vol. 127, pp. 160–168, 2019. [Online]. Available: https://doi.org/10.1016/j.jpdc.2018.06.008

[9] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Softw. Pract. Exp.*, vol. 50, no. 5, pp. 489–502, 2020. [Online]. Available: https://doi.org/10.1002/spe.2685

[10] M. Tao, K. Ota, and M. Dong, "Foud: Integrating fog and cloud for 5g-enabled V2G networks," *IEEE Netw.*, vol. 31, no. 2, pp. 8–13, 2017. [Online]. Available: https://doi.org/10.1109/MNET.2017.1600213NM

[11] D. Lu, Y. Qu, F. Wu, H. Dai, C. Dong, and G. Chen, "Robust server placement for edge computing," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, May 18-22, 2020*. IEEE, 2020, pp. 285–294. [Online]. Available: https://doi.org/10.1109/IPDPS47924.2020.00038

[12] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016. [Online]. Available: https://doi.org/10.1109/TNET.2015.2487344

[13] I. Pelle, F. Paolucci, B. Sonkoly, and F. Cugini, "Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 9, pp. 2849–2863, 2021. [Online]. Available: https://doi.org/10.1109/JSAC.2021.3064655

[14] H. Yin, X. Zhang, H. H. Liu, Y. Luo, C. Tian, S. Zhao, and F. Li, "Edge provisioning with flexible server placement," *IEEE Trans. Parallel Distributed Syst.*, vol. 28, no. 4, pp. 1031–1045, 2017. [Online]. Available: https://doi.org/10.1109/TPDS.2016.2604803

[15] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 06, pp. 725–737, 2017. [Online]. Available: https://doi.org/10.1109/TCC.2015.2449834

[16] S. K. Kasi, M. K. Kasi, K. Ali, M. Raza, H. Afzal, A. Lasebae, B. Naeem, S. ul Islam, and J. J. P. C. Rodrigues, "Heuristic edge server placement in industrial internet of things and cellular networks," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10 308–10 317, 2021. [Online]. Available: https://doi.org/10.1109/JIOT.2020.3041805

[17] F. Guo, B. Tang, and J. Zhang, "Mobile edge server placement based on meta-heuristic algorithm," *J. Intell. Fuzzy Syst.*, vol. 40, no. 5, pp. 8883–8897, 2021. [Online]. Available: https://doi.org/10.3233/JIFS-200933

[18] X. Chen, W. Liu, J. Chen, and J. Zhou, "An edge server placement algorithm in edge computing environment," in *12th International Conference on Advanced Infocomm Technology, ICAIT 2020, Taipa, Macao, November 23-25, 2020*. IEEE, 2020, pp. 85–89. [Online]. Available: https://doi.org/10.1109/ICAIT51223.2020.9315526

[19] H. Yin, X. Zhang, H. H. Liu, Y. Luo, C. Tian, S. Zhao, and F. Li, "Edge provisioning with flexible server placement," *IEEE Trans. Parallel Distributed Syst.*, vol. 28, no. 4, pp. 1031–1045, 2017. [Online]. Available: https://doi.org/10.1109/TPDS.2016.2604803

[20] B. Li, P. Hou, H. Wu, and F. Hou, "Optimal edge server deployment and allocation strategy in 5g ultra-dense networking environments," *Pervasive Mob. Comput.*, vol. 72, p. 101312, 2021. [Online]. Available: https://doi.org/10.1016/j.pmcj.2020.101312

[21] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, 2017. [Online]. Available: https://doi.org/10.1109/TCC.2015.2449834

[22] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 2, pp. 787–796, 2018. [Online]. Available: https://doi.org/10.1109/TNSM.2018.2816263

[23] N. Mohan, A. Zavodovski, P. Zhou, and J. Kangasharju, "Anveshak: Placing edge servers in the wild," in *Proceedings of the 2018 Workshop on Mobile Edge Communications, MECOMM@SIGCOMM 2018, Budapest, Hungary, August 20, 2018*. ACM, 2018, pp. 7–12. [Online]. Available: https://doi.org/10.1145/3229556.3229560

[24] S. Mondal, G. Das, and E. Wong, "CCOMPASSION: A hybrid cloudlet placement framework over passive optical access networks," in *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*. IEEE, 2018, pp. 216–224. [Online]. Available: https://doi.org/10.1109/INFOCOM.2018.8485846

[25] S. Zhao, X. Zhang, P. Cao, and X. Wang, "Design of robust and efficient edge server placement and server scheduling policies," in *29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021*. IEEE, 2021, pp. 1–7. [Online]. Available: https://doi.org/10.1109/IWQOS52092.2021.9521290

[26] B. Cao, S. Fan, J. Zhao, S. Tian, Z. Zheng, Y. Yan, and P. Yang, "Large-scale many-objective deployment optimization of edge servers," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3841–3849, 2021. [Online]. Available: https://doi.org/10.1109/TITS.2021.3059455

[27] S. Lu, J. Wu, and Z. Fang, "High-elasticity virtual cluster placement in multi-tenant cloud data centers," in *21st IEEE International Conference on High Performance Computing and Communications, HPCC 2019, Zhangjiajie, China, August 10-12, 2019*. IEEE, 2019, pp. 996–1003. [Online]. Available: https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00143

[28] H. H. Sinky, B. Khalfi, B. Hamdaoui, and A. Rayes, "Adaptive edge-centric cloud content placement for responsive smart cities," *IEEE Netw.*, vol. 33, no. 3, pp. 177–183, 2019. [Online]. Available: https://doi.org/10.1109/MNET.2019.1800137

[29] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distributed Comput.*, vol. 127, pp. 160–168, 2019. [Online]. Available: https://doi.org/10.1016/j.jpdc.2018.06.008

[30] Y. Zhang, P. Li, Z. Zhang, B. Bai, G. Zhang, W. Wang, B. Lian, and K. Xu, "Autosight: Distributed edge caching in short video network," *IEEE Netw.*, vol. 34, no. 3, pp. 194–199, 2020. [Online]. Available: https://doi.org/10.1109/MNET.001.1900345

[31] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proceedings IEEE INFOCOM 2001, The Conference on Computer Communications, Anchorage, Alaska, USA, April 22-26, 2001*. IEEE Comptuer Society, 2001, pp. 1587–1596. [Online]. Available: https://doi.org/10.1109/INFCOM.2001.916655

[32] J. Sahoo, M. A. Salahuddin, R. H. Glitho, H. Elbiaze, and W. Ajib, "A survey on replica server placement algorithms for content delivery networks," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 2, pp. 1002–1026, 2017. [Online]. Available: https://doi.org/10.1109/COMST.2016.2626384

[33] Y. Zhang, P. Li, Z. Zhang, C. Zhang, W. Wang, Y. Ning, and B. Lian, "Graphinf: A gcn-based popularity prediction system for short video networks," in *Web Services - ICWS 2020 - International Conference on Web Services, Honolulu, HI, USA, September 18-20, 2020, Proceedings*, vol. 12406. Springer, 2020, pp. 61–76. [Online]. Available: https://doi.org/10.1007/978-3-030-59618-7_5