Online Service Caching and Routing at the Edge with Unknown Arrivals

Siqi Fan, I-Hong Hou
Texas A&M University
College Station, USA
{siqifan, ihou}@tamu.edu

Van Sy Mai, Lotfi Benmohamed
National Institute of Standards and Technology
Gaithersburg, USA
{vansy.mai, lotfi.benmohamed}@nist.gov

Abstract—**This paper studies a problem of jointly optimizing two important operations in mobile edge computing without knowing future requests, namely service caching, which determines which services to be hosted at the edge, and service routing, which determines which requests to be processed locally at the edge. We aim to address several practical challenges, including limited storage and computation capacities of edge servers and unknown future request arrival patterns. To this end, we formulate the problem as an online optimization problem, in which the objective function includes costs of forwarding requests, processing requests, and reconfiguring edge servers. By leveraging a natural timescale separation between service routing and service caching, namely, the former happens faster than the latter, we propose an online two-stage algorithm and its randomized variant. Both algorithms have low complexity, and our fractional solution achieves sublinear regret. Simulation results show that our algorithms significantly outperform other state-of-the-art online policies.**

Index Terms—**Edge Network, Online Optimization, Service Caching, Service Routing**

I. INTRODUCTION

A growing challenge for mobile computing is the proliferation of data/computation-intensive and delay-sensitive applications, such as cognitive assistance and augmented reality (AR). On the one hand, running these applications completely within mobile devices may be infeasible due to the limited computation, storage, and battery capacity of such devices. On the other hand, offloading computation tasks of these applications to remote data centers may result in excessive end-to-end latency and hence poor user experience.

Such a dilemma has given rise to the popularity of mobile edge computing [1], [2]. In mobile edge computing, edge servers are deployed close to wireless base stations. These servers can host some popular services and process the corresponding computation tasks directly without having to forward them to remote data centers. Due to their close proximity to end users, edge servers are able to provide these services with much lower latency.

Despite the obvious advantage of mobile edge computing, there remain multiple important challenges that need to be addressed. First, edge servers can often host (or cache) a small number of services, and installing new services is typically time-consuming and expensive since it involves downloading all necessary data from remote data centers and setting up appropriate virtual machines or containers. Second, edge servers

usually have limited computation power, and hence requests will suffer queuing delays. So, the edge server needs to decide whether to process a request locally or not, even it has already cached the corresponding service. Third, mobile users generate requests for services in arbitrary and typically time-varying patterns, which are hard to learn. Thus, edge servers must decide which services to cache and which requests to process without knowledge of future requests.

Most existing works only focus on one or two challenges above. Some studies only address the online caching problem with unknown request arrival patterns. For example, Paschos *et al.* [3], [4], and Gao *et al.* [5] propose online algorithms with sublinear regret based on the online gradient ascent and bandit learning method. Zhao *et al.* [6] address the installation cost and analyze the competitive ratio of their algorithm. These studies fail to take the limited computation power of edge servers into account. Some other papers consider joint designs of service caching and request processing by explicitly address limits on both storage and computation power. For instance, Li *et al.* [7] and Xu *et al.* [8] propose online algorithms for optimizing service caching and request routing based on a Lyapunov optimization framework. A weakness of these caching and routing solutions is that they assume that the request arrival patterns are predictable or follow a certain stationary random process.

In this paper, we aim to address all three challenges and minimize a combination of the queuing latency, forwarding latency, and installation costs. By leveraging the natural timescale separation between service caching and service routing, we formulate the problem into a two-stage online optimization problem without knowledge of future requests.

To solve this problem, we propose a low-complexity two-stage online policy and its randomized variant. The two-stage online policy consists of two parts: the first part is a low-complexity algorithm that finds not only the optimal service routing decision but also the gradient of the current service caching decision, despite that neither of them have closed-form expressions, and the second part employs online projected gradient descent to update the service caching decisions. We further design its randomized variant to make the probabilistic caching solution from the two-stage online policy implementable. We theoretically prove that our two-stage online policy achieves sublinear regret, while the randomized one at

most triples the installation cost.

Our online algorithms are evaluated through simulations under various scenarios. We compare them against three other algorithms, including an offline policy that knows overall request popularity in advance. Results show that our algorithms perform much better than other online algorithms and perform virtually the same as the offline policy.

The rest of the paper is organized as follows. Section II introduces our system model and problem formulation. Section III presents our two-stage online algorithm for obtaining fractional solutions with sublinear regret. Section IV includes a randomized variant of the algorithm that ensures integer solutions for the service caching problem. Section V shows our simulation results under a variety of scenarios. Finally, Section VI concludes the paper.

II. System Model

A. System Overview

We consider an edge system with a backhaul connection. This edge system includes multiple clients, an edge server, and remote data centers. Clients generate requests for different services according to some unknown and unpredictable patterns, and then send these requests to the edge server. The edge server may *cache* some services and process some requests for these services locally, while forwarding remaining requests to remote data centers. Requests processed at the edge encounter a *computation latency* due to the limited computation capacity of the edge server, while requests forwarded to remote data centers encounter a *forwarding latency* due to network latency.

B. Service Caching and Processing

We assume that time is slotted, and the system runs for T time slots. The duration of a time slot is chosen so that, in any given time slot, the patterns for the service requests (originating from different clients) remain roughly the same.

We use N to denote the total number of different services. Let $x_{n,t} \in \{1, 0\}$ be a binary decision variable that indicates whether the edge server caches service n in time slot t, and let $X_t := [x_{1,t}, x_{2,t}, \ldots, x_{N,t}]$. Since the edge server has limited storage capacity, we assume that the edge server can cache at most Z services, i.e., $\sum_{n=1}^{N} x_{n,t} \leq Z, \forall t$. We call the problem of determining X_t the *service caching problem*.

Caching a new service at the edge can be a costly process, which typically involves downloading codes and databases and setting up virtual machines or containers. Thus, we assume that the edge server must decide X_t before time slot t.

At the beginning of time slot t, the edge server observes the requests from clients and calculates the request arrival rates. We use $\lambda_{n,t}$ to denote the number of requests for service n in time slot t, and let $\Lambda_t := [\lambda_{1,t}, \lambda_{2,t}, \ldots, \lambda_{N,t}]$. We assume that an upper bound W on the total arrival rate is known, that is, $\sum_{n=1}^{N} \lambda_{n,t} \leq W, \forall t$.

During each time slot t, the edge server needs to decide which requests to be processed locally. Due to the limited computation power of the edge server, it may not be desirable to process all requests for services that it caches. For a service n, the edge server will process a fraction $y_{n,t} \in [0, 1]$ of the number of requests locally, and forward the remaining $(1 - y_{n,t})$ portion of the requests to the data center. Since the edge server can only process requests whose corresponding services have already been cached at the edge, we require that $y_{n,t} \leq x_{n,t}, \forall n$.

Let $Y_t := [y_{1,t}, y_{2,t}, \ldots, y_{N,t}]$. We call the problem of determining Y_t the *service routing problem*. Since the edge server can adjust service routing in real time, we consider that the edge server determines Y_t after it observes Λ_t.

C. Cost and Problem Formulation

The goal of the edge server is to minimize the total cost of the system, which consists of *latency cost* and *installation cost*, by jointly optimizing X_t and Y_t.

First, the latency cost refers to the total latency experienced by all requests. In the system, when a request is forwarded to the remote data center, it experiences a forwarding latency, which is denoted as d_n for service n. When a request is processed at the edge, it experiences a computation latency due to the limited computation power of the edge server. It is reasonable to assume that the per-request computation latency at the edge depends on the total computation load, and can be described by a convex, increasing, and differentiable function $C(\cdot)$ with $0 \leq C(0) \leq d_n, \forall n$ and $\lim_{s \to \infty} C(s) = \infty$. Since the total computation load at the edge server is $\sum_{n=1}^{N} \lambda_{n,t} y_{n,t}$ in time slot t, the total latency of all requests can be written as

$$L_t(Y_t) := \sum_{n=1}^{N} \lambda_{n,t} y_{n,t} C\Big(\sum_{m=1}^{N} \lambda_{m,t} y_{m,t}\Big) + \sum_{n=1}^{N} \lambda_{n,t}(1 - y_{n,t}) d_n.$$

The goal of the edge server at each time slot t is to minimize this latency cost given the current caching decision X_t, i.e., to solve

$$G_t(X_t) := \min_{Y_t} \quad L_t(Y_t), \tag{1}$$

$$\text{s.t.} \quad 0 \leq y_{n,t} \leq x_{n,t}, \quad \forall n. \tag{2}$$

Second, the installation cost refers to the operation cost incurred when the edge server caches new services. For simplicity, we assume that caching every new service incurs a cost of β. Hence, the total installation cost is $\beta \sum_{t=1}^{T} \|X_t - X_{t-1}\|_+$, where $\|X_t\|_+ = \sum_{n=1}^{N} \max\{x_{n,t}, 0\}$. As a result, the sequence of determining variables and receiving cost is illustrated in Fig. 1, and the total cost over T time slots can be written as $\sum_{t=1}^{T} (G_t(X_t) + \beta \|X_t - X_{t-1}\|_+)$.
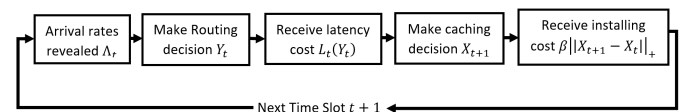

Fig. 1. Process of online service caching and routing.

Here, it is important to note that the service caching problem and the service routing problem operate on different timescales. The edge server needs to decide X_t in time slot

$t-1$, without any knowledge about Λ_t. In contrast, the edge server can decide the value of Y_t after observing the first few requests in time slot t to estimate the arrival rate Λ_t.

The edge server aims to find $[X_1, X_2, \ldots, X_T]$ and $[Y_1, Y_2, \ldots, Y_T]$ that minimize the total cost subject to all the constraints described above. However, since this is a mixed integer problem due to binary caching decisions $x_{n,t} \in \{0, 1\}$, we first relax this constraint and allow $x_{n,t}$ to be any real number in $[0, 1]$. Under this relaxation, $x_{n,t}$ can be interpreted as the probability of caching service n at time t, and the (offline) problem of minimizing the total cost becomes:

$$\min_{[X_t]} \quad \sum_{t=1}^{T} \Big(G_t(X_t) + \beta \|X_t - X_{t-1}\|_+ \Big), \qquad (3)$$

$$\text{s.t.} \quad 0 \le x_{n,t} \le 1, \quad \forall n, \forall t, \qquad (4)$$

$$\sum_{n=1}^{N} x_{n,t} \le Z, \quad \forall t. \qquad (5)$$

While the above problem is a convex optimization problem, solving it requires the knowledge of all request arrival rates, that is, all Λ_t, in advance. In practice, however, the edge server needs to make service caching decision X_t without the knowledge of future arrival rates, hence the need for an *online algorithm*.

The performance of an online algorithm that does not have any knowledge about future arrivals is evaluated by comparing it against an offline policy that knows future arrivals. Let us consider an offline policy that knows the overall popularity of all services, i.e., $\sum_t \lambda_{n,t}$ for all $n \in [1, N]$, but not individual $\lambda_{n,t}$. In this case, the optimal static caching policy will cache Z services with largest $d_n \sum_t \lambda_{n,t}$ at all times. The assumption for this offline optimal static policy is widely used in caching literature; see, e.g., [9]. Let $X^o := [x_1^o, x_2^o, \ldots, x_N^o]$ be the service caching decision of this offline policy. Note that, since we assume that the edge server can adjust service routing in real time, the offline policy can determine an optimal routing decision $[Y_t^o]$ that minimizes the latency cost $L_t(Y_t)$ in each time slot t while satisfying the constraint $y_{n,t}^o \le x_n^o, \forall n$.

Let $\hat{X} := [\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_T]$ and $\hat{Y} := [\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_T]$ be solutions produced by an online algorithm ξ, then we define the regret of ξ as the difference between its cost and the cost of the optimal static offline policy:

$$Reg(\xi) := \sum_{t=1}^{T} \Big(L_t(\hat{Y}_t) - L_t(Y_t^o) + \beta \|\hat{X}_t - \hat{X}_{t-1}\|_+ \Big).$$

It should be noted that the offline policy doesn't incur installation costs since it uses a fixed caching decision.

The goal of this paper is to find an online algorithm with provably small regret under any sequence of arrival rates.

III. ONLINE ALGORITHM

In this section, we present a two-stage online algorithm that aims to jointly optimize the service caching and routing decisions asymptotically. After arrival rates are revealed in each time slot, the first stage computes routing decisions

and produces necessary parameters for the next stage, which updates caching decisions for the next time slot.

A. Optimal Routing

In this subsection, we consider the service routing problem at time t described in (1)–(2), given the current caching decision X_t and arrival rates Λ_t. Solving this problem with a general convex optimization solver may, however, incur high complexity. Surprisingly, we show below that there exists an $O(N)$ algorithm that not only solves this problem but also provides a subgradient $\nabla G_t(X_t)$, which is important for dealing with the service caching problem in the second stage.

The main idea of our algorithm is to leverage the special structure in $L_t(Y_t)$. Let

$$J_t(Y_t) := C\Big(\sum_{n=1}^{N} \lambda_{n,t} y_{n,t}\Big) + \sum_{n=1}^{N} \lambda_{n,t} y_{n,t} C'\Big(\sum_{m=1}^{N} \lambda_{m,t} y_{m,t}\Big).$$

Then, we have $\frac{1}{\lambda_{n,t}} \frac{\partial L_t(Y_t)}{\partial y_{n,t}} = J_t(Y_t) - d_n$, which corresponds to the marginal benefit of processing one more request for service n at the edge. Sorting all services so that $d_1 \ge d_2 \ge \cdots \ge d_N$, then we have $\frac{1}{\lambda_{1,t}} \frac{\partial L_t(Y_t)}{\partial y_{1,t}} \le \frac{1}{\lambda_{2,t}} \frac{\partial L_t(Y_t)}{\partial y_{2,t}} \le \cdots \le \frac{1}{\lambda_{N,t}} \frac{\partial L_t(Y_t)}{\partial y_{N,t}}$. Based on this observation, we design Algorithm 1 shown below.

Algorithm 1 `ServiceRouting`

Input: $d_1 \ge d_2 \ge \ldots \ge d_N$, X_t, Λ_t
Initialize: $Y_t \leftarrow 0$
1: **for** $n = 1, 2, \ldots, N$ **do**
2:     **if** $J_t(Y_t) - d_n < 0$ **then**
3:         $y_{n,t} \leftarrow x_{n,t}$
4:     **if** $J_t(Y_t) - d_n > 0$ **then**
5:         choose $y_{n,t} \in [0, x_{n,t}]$ s.t. $J_t(Y_t) - d_n = 0$
6: **for** $n = 1, 2, \ldots, N$ **do**
7:     **if** $J_t(Y_t) \le d_n$ **then**
8:         $\nu_n \leftarrow \lambda_{n,t}(d_n - J_t(Y_t))$, $\mu_n \leftarrow 0$
9:     **else**
10:       $\nu_n \leftarrow 0$, $\mu_n \leftarrow \lambda_{n,t}(J_t(Y_t) - d_n)$
Output: $Y_t, \nabla G_t(X_t) \leftarrow [-\nu_1, -\nu_2, \ldots, -\nu_N]$

Theorem 1: Algorithm 1 produces an optimal solution for the routing problem (1)–(2) and a subgradient $\nabla G_t(X_t)$ is given by

$$\frac{\partial G_t(X_t)}{\partial x_{n,t}} = -\nu_n = \begin{cases} \lambda_{n,t}(J_t(Y_t) - d_n), & \text{if } y_{n,t} = x_{n,t}, \\ 0, & \textbf{otherwise}. \end{cases}$$

Proof: First, it can be verified that Y_t, ν_n and μ_n produced by Algorithm 1 satisfy the KKT conditions of problem (1)–(2), i.e.,

$$\lambda_{n,t}(J_t(Y_t) - d_n) - \mu_n + \nu_n = 0, \forall n, \qquad (6)$$

$$\nu_n(y_{n,t} - x_{n,t}) = 0, \mu_n(-y_{n,t}) = 0, \forall n, \qquad (7)$$

$$\mu_n \ge 0, \ \nu_n \ge 0, \forall n, \qquad (8)$$

where ν_n and μ_n are Lagrange multipliers associated with the constraints in (2). Since the problem is convex, it follows that Y_t is an optimal solution.

Second, it follows from [10, §5.6] that G_t is convex in X_t and that $\frac{\partial G_t(X_t)}{\partial x_{n,t}} = -\nu_n, \forall n = 1, \ldots, N$. This completes the proof. ∎

B. Online Service Caching

For service caching, we adopt the online gradient descent method with lazy projection in [11], where the update step at time t is given in Algorithm 2 below. Here, $\nabla G_t(X_t)$ is the subgradient calculated by Algorithm 1, η is the step size, and $\theta_t = [\theta_{1,t}, \theta_{2,t}, \ldots, \theta_{N,t}]$ is an internal vector with $\theta_1 = \mathbf{0}$.

Algorithm 2 `ServiceCaching`

Input: $\theta_t, \nabla G_t(X_t), \eta$
1: $\theta_{t+1} \leftarrow \theta_t - \nabla G_t(X_t)$
2: $X_{t+1} \leftarrow$ the Euclidean projection of $\eta\theta_{t+1}$ onto the set $\{X \in \mathbb{R}^N \mid 0 \leq x_n \leq 1, \sum_{n=1}^N x_n \leq Z\}$
Output: X_{t+1}, θ_{t+1}

As a result, combining both Algorithms 1 and 2 yields an online service caching and routing algorithm (OCR) shown in Algorithm 3 below.

Algorithm 3 Online service Caching and Routing (OCR)

Initialize: $\eta, \theta_1 \leftarrow 0, X_1$
1: **for** $t = 1, 2, ..., T$ **do**
2: $\quad Y_t, \nabla G_t(X_t) \leftarrow$ `ServiceRouting`(X_t, Λ_t)
3: $\quad X_{t+1}, \theta_{t+1} \leftarrow$ `ServiceCaching`$(\theta_t, \nabla G_t(X_t), \eta)$

Next we show that OCR achieves a sublinear regret.

Theorem 2: Let $\eta = O(\frac{1}{\sqrt{T}})$. Then, $Reg(OCR) = O(\sqrt{T})$.

Proof: Since Algorithm 3 can be viewed as applying online gradient descent with lazy projection to the problem in (3) with objective function $G_t(X_t)$ (i.e., without the installation cost), it follows from [11, Corollary 2.17] that the regret (in terms of $G_t(X_t)$) is $O(\sqrt{T})$ when $\eta = O(\frac{1}{\sqrt{T}})$, provided that ∇G_t is bounded. In our case, the boundedness holds because

$$\|\nabla G_t(X_t)\|_2^2 = \sum_{n=1}^N \nu_n^2 \leq \sum_{n=1}^N \lambda_{n,t}^2 d_n^2 \leq W^2 \max_i d_i^2, \quad (9)$$

where we have used Theorem 1 and the fact that $\sum_{n=1}^N \lambda_{n,t}^2 \leq \left(\sum_{n=1}^N \lambda_{n,t}\right)^2 = W^2$.

It remains to show that $\beta \sum_{t=1}^T \|X_t - X_{t-1}\|_+ = O(\sqrt{T})$. To this end, note that $\|X_t - X_{t-1}\|_+ \leq \|X_t - X_{t-1}\|_1 \leq \sqrt{N}\|X_t - X_{t-1}\|_2 \leq \sqrt{N}\|\eta\nabla G_t(X_t)\|_2$, where the last inequality follows from Algorithm 2 and the nonexpansiveness property of Euclidean projections. Next, using (9) and the fact that $\eta = O(\frac{1}{\sqrt{T}})$, we have $\sum_{t=1}^T \beta\|X_t - X_{t-1}\|_+ \leq \sum_{t=1}^T \eta\beta\sqrt{N}\|\nabla G_t(X_t)\|_2 \leq T\eta\beta\sqrt{N}\max_i d_i = O(\sqrt{T})$.

Thus, we conclude that $Reg(OCR) = O(\sqrt{T})$. ∎

Finally, we analyze the complexity of Algorithm 3. It can be seen that the bottleneck is the projection step in the line 2 of Algorithm 2.

We consider the following steps for finding the projection of $\eta\theta_{n,t+1}$ onto the set $\{X \in \mathbb{R}^N \mid 0 \leq x_n \leq 1, \sum_{n=1}^N x_n \leq Z\}$. Let X' be the vector of x'_n where $x'_n = \min\{1, \max\{0, \eta\theta_{n,t+1}\}\}$. If $\sum_{n=1}^N x'_n \leq Z$, then X' is the projection. Otherwise, the projection, denoted by X^*,

must have $\sum_{n=1}^N x_n^* = Z$. Then, we can employ the algorithm in [12], which has complexity $O(N^2)$, to obtain X^*. Hence, the overall complexity of Algorithm 3 is $O(N^2)$ per time slot.

IV. RANDOMIZED ALGORITHM FOR SERVICE CACHING

The online algorithm for finding X_t as proposed in Algorithm 2 may produce fractional solutions, which can be interpreted as the probability that the edge server caches each service. In this section, we propose a randomized algorithm that satisfies this probability interpretation while guaranteeing a provably small installation cost.

A. Randomized Algorithm

The basic idea of our randomized algorithm is to simultaneously maintain K sample paths, where each sample path represents a probability mass of $\frac{1}{K}$. We then quantize each $x_{n,t}$ into a multiple of $\frac{1}{K}$. Specifically, let X_t^Q be the quantized version of X_t, we then require that $K x_{n,t}^Q$ to be a non-negative integer and $\sum_n x_{n,t}^Q \leq Z$.

Let $r_{k,n,t}$ be the indicator function that service n is cached at the edge at time t in the sample path k. Let $R_{k,t}$ be the vector $[r_{k,1,t}, r_{k,2,t}, \ldots]$. In every time slot t, our randomized algorithm receives X_t^Q from Algorithm 2. Then, it constructs $R_{k,t}$ based on X_t^Q and $R_{k,t-1}$ to ensure three properties: First, the probability of caching service n is indeed $x_{n,t}^Q$, that is, $\sum_{k=1}^K r_{k,n,t} = K x_{n,t}^Q$. Second, the storage capacity constraint is satisfied for all sample paths, that is, $\sum_n r_{k,n,t} \leq Z, \forall k$. Third, the expected installation cost, which can be expressed as $\frac{1}{K}\sum_k \|R_{k,t} - R_{k,t-1}\|_+$, is bounded. Let $\Delta_t := [\delta_{1,t}, \delta_{2,t}, \ldots, \delta_{N,t}]$ be the difference between X_t^Q and X_{t-1}^Q. Algorithm 4 shows the complete randomized algorithm, including decisions on service caching and routing.

Algorithm 4 Randomized Online service Caching and Routing (ROCR)

Initialize: K, $R_{k,1} \leftarrow 0, \forall k, \eta, \theta_1 \leftarrow 0$
1: Choose k^* uniformly from $\{1, 2, \ldots, K\}$.
2: **for** $t = 1, 2, ..., T$ **do**
3: $\quad$ Observe Λ_t.
4: $\quad Y_t, \nabla G_t(X_t) \leftarrow$ `ServiceRouting`$(R_{k^*,t}, \Lambda_t)$.
5: $\quad X_{t+1}^Q, \theta_{t+1} \leftarrow$ `ServiceCaching`$(\theta_t, \nabla G_t(X_t), \eta)$.
6: $\quad R_{k,t+1} \leftarrow R_{k,t}, \forall k$.
7: $\quad \Delta_{t+1} \leftarrow X_{t+1}^Q - X_t^Q$.
8: $\quad$ **for** $n = 1, 2, \ldots, N$ **do**
9: $\quad\quad$ **if** $\delta_{n,t+1} > 0$ **then**
10: $\quad\quad\quad$ Randomly choose $K\delta_{n,t+1}$ sample paths with $r_{k,n,t+1} = 0$, and set $r_{k,n,t+1} = 1$ for them.
11: $\quad\quad$ **else if** $\delta_{n,t} < 0$ **then**
12: $\quad\quad\quad$ Randomly choose $|K\delta_{n,t+1}|$ sample paths with $r_{k,n,t+1} = 1$, and set $r_{k,n,t+1} = 0$ for them.
13: $\quad$ **while** $\exists \hat{k}$ such that $\sum_n r_{\hat{k},n,t+1} > Z$ **do**
14: $\quad\quad$ Find one sample path k' with $\sum_n r_{k',n,t+1} < Z$.
15: $\quad\quad$ Find a service $\hat{n}$ with $r_{\hat{k},\hat{n},t+1} = 1, r_{k',\hat{n},t+1} = 0$.
16: $\quad\quad$ Set $r_{\hat{k},\hat{n},t+1} = 0$ and $r_{k',\hat{n},t+1} = 1$.
17: $\quad$ Cache all services with $r_{k^*,n,t+1} = 1$.

By the design of Algorithm 4, we obviously have the first two properties. We show below that Algorithm 4 also enjoys a provably small expected installation cost.

B. Performance Analysis

First, we consider the influence of Algorithm 4 on the installation cost, which is shown below.

Theorem 3: The expected installation cost at each time slot in Algorithm 4 is at most $3\beta\|X_t^Q - X_{t-1}^Q\|_+$.

Proof: As the installation cost only happens when we increase $r_{k,n,t}$, we aim to bound the increase in $r_{k,n,t}$. Under Algorithm 4, $r_{k,n,t}$ can be changed either in lines 9–12 or in lines 14–16. In lines 9–12, the total increase is $K\|X_t^Q - X_{t-1}^Q\|_+$. Moreover, every change in lines 9–12 can result in at most two changes in lines 14–16. Hence, the total increase in lines 14–16 is at most $2K\|X_t^Q - X_{t-1}^Q\|_+$.

Thus, the maximum increase in Algorithm 4 is $3K\|X_t^Q - X_{t-1}^Q\|_+$ over all sample paths. Since each sample path represents a probability mass of $\frac{1}{K}$, the expected installation cost is at most $3\beta\|X_t^Q - X_{t-1}^Q\|_+$. ∎

Next, we analyze the complexity of Algorithm 4. Since $\sum_n X_t^Q \le Z$ and $\sum_n X_{t-1}^Q \le Z$, at most KZ variables will be increased to 1 and at most KZ variables will be decreased to 0 in Steps 10–12. This is a total of $O(KZ)$ changes. To implement the while loop in Steps 14–16, we can first divide all sample paths into three groups: those with $\sum_n r_{\hat{k},n,t} > Z$, those with $\sum_n r_{\hat{k},n,t} = Z$, and those with $\sum_n r_{\hat{k},n,t} < Z$. Then, Step 14 is an $O(1)$ operation. Step 16 takes $O(N)$ time. We note that each increase in Steps 10–12 will result in at most one iteration of the while loop in Steps 14–16. Hence, steps 14–16 will be executed at most KZ times and the overall complexity of this while loop is $O(KZN)$. Thus, the complexity of Algorithm 4 is $O(\max\{KZN, N^2\})$ per time slot.

V. SIMULATION RESULTS

In this section, we conduct various simulations to evaluate the performance of our algorithms OCR and ROCR.

A. Setup

We conduct experiments on following two datasets:

- The first dataset is a synthetic dataset, following a random replacement model in [3], [13] with $N = 10^3$ and $T = 10^4$. In this dataset, all requests follow a Zipf distribution, while the ranking of services frequently changes according to Table 2 in [13].
- The second dataset is based on the Google trace data from [14], containing a sequence of different service requests. This dataset includes more than three million requests for $N = 9,218$ services within a seven-hour timespan. As time is slotted in the trace data by 300 seconds, which is a large jump, we divide each interval into 300 different parts with an equal number of requests following the original sequence. In this dataset, the popularity of requests in one time slot changes fast, while some services are very popular over the whole time period.

Considering the queuing delay, we assume that the edge server operates like a $M/M/1$ queuing system [15] with service rate ϕ, i.e., $C(\sum_{i=1}^N y_{i,t}) = \frac{1}{\phi - \sum_{i=1}^N y_{i,t}}$.

The system parameters are shown in Table I, where the values of forwarding latency, service rate, and cache limit follow the parameters of services and base stations in [8].

TABLE I
SYSTEM PARAMETERS

Parameter	d_n (sec/request)	ϕ (request/sec)	Z	K	η
Value	$[2,4]$	$[20,100]$	$[2,10]$	10^2	0.05

Throughout the evaluation, we compare ROCR and OCR with the following baseline approaches:

- OGA (Online Gradient Ascent [3]): In each time slot, it uses $[\lambda_{1,t}d_1, \lambda_{2,t}d_2, \ldots, \lambda_{N,t}d_N]$ as the gradient for the service caching problem. Since OGA does not consider the routing procedure, we apply our routing policy in this algorithm to obtain its best performance. OGA produces fractional X_t and its cost is based on this fractional X_t.
- OFF (Offline Policy): This is the optimal static offline policy defined in Sec. II. It caches the same Z services with the largest $\sum_{t=1}^T \lambda_{n,t}d_n$ in all time slots and applies optimal routing decisions.
- OREO: (Online seRvice caching for mobile Edge cOmputing [8]): This algorithm jointly optimizes service caching and routing decisions with energy and cost constraints. In the context of this work, all energy and cost constraints in [8] are relaxed to be infinite. As suggested by [8], we let the arrivals of the current time slot be the prediction for the next time slot and use the Gibbs sampling method with parameter $\tau = 10^{-2}$ to update caching decisions.

We evaluate the performance of all five algorithms with different values of edge server caching limit Z, service rate ϕ, and installation cost parameter β. We choose $\phi = 60$, $Z = 6$, and $\beta = 100$ if they are not specified. In addition, we present the regret of all four online algorithms in each time slot.

B. Evaluation Results

The simulation results for two scenarios are shown in Fig. 2 and Fig. 3, and we can obtain several important observations.

First, our ROCR significantly outperforms OREO in all settings. Though ROCR and OREO all jointly optimize service caching and routing, OREO assumes request arrival patterns are predictable and uses Gibbs sampling for cache updates, which causes massive installation cost and the surprising cost increment when we increase the cache size Z. Second, ROCR also outperforms OGA in all scenarios. While both algorithms are based on online gradient methods, ROCR can achieve better performance because it explicitly considers the processing latency and avoids the redundant cache changes when the edge server cannot process requests for popular services locally. Observations above show that any online algorithm for edge computing needs to address both memory and computation power constraints of edge servers as well as the challenge of unknown future requests.
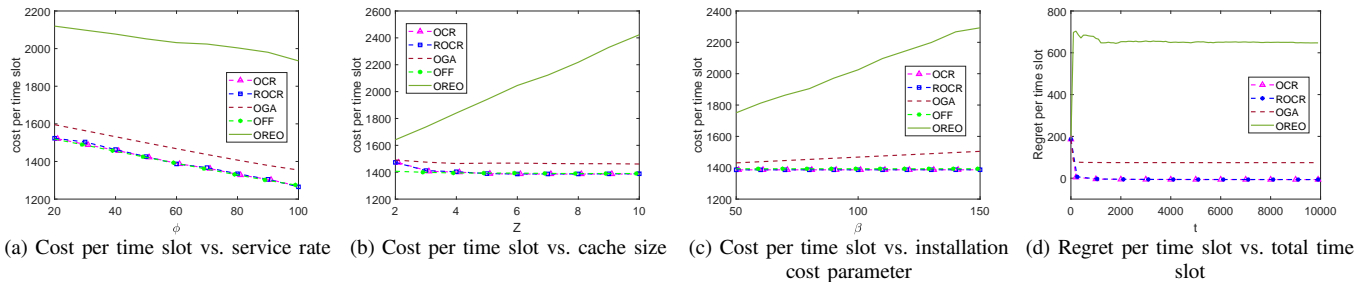
| (a) Cost per time slot vs. service rate | (b) Cost per time slot vs. cache size | (c) Cost per time slot vs. installation cost parameter | (d) Regret per time slot vs. total time slot |

Fig. 2. Simulation results using synthetic data.

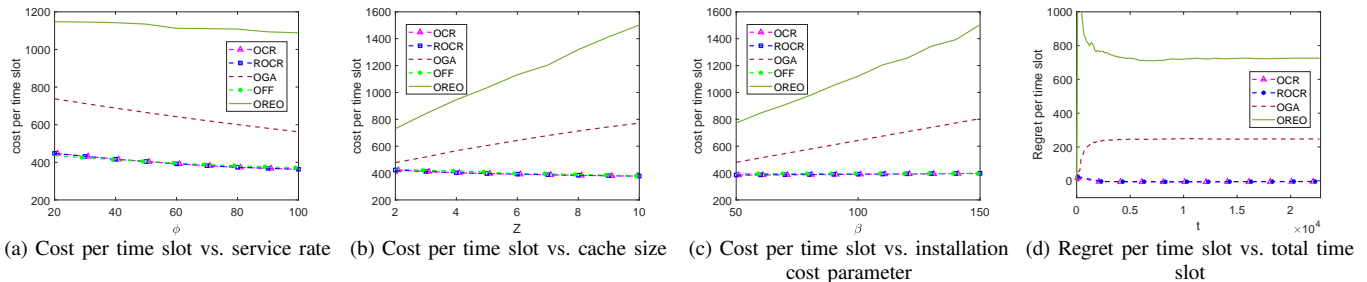| (a) Cost per time slot vs. service rate | (b) Cost per time slot vs. cache size | (c) Cost per time slot vs. installation cost parameter | (d) Regret per time slot vs. total time slot |

Fig. 3. Simulation results using the Google trace data.

Third, our OCR has virtually the same performance as the optimal static offline policy and achieves nearly zero regrets, which is consistent with our analysis.

Finally, we note that ROCR and OCR have very similar performances in all cases. OCR produces fractional solutions for the service caching problem, and then ROCR transforms such fractional solutions into randomized solutions with integer solutions on every sample path. As discussed in Section IV, by carefully choosing which services to host at the edge on every sample path, ROCR is able to incur an installation cost that is at most three times larger than that of OCR. Our simulation results further show that the overall costs of ROCR and OCR are almost identical in practical scenarios.

VI. CONCLUSION

This paper studies the problem of service caching and routing without any knowledge about future requests. Motivated by a practical timescale separation, we formulate this problem as a two-stage online optimization problem that jointly considers the storage and computation constraints of the edge server, as well as the installation cost. We propose a low-complexity online algorithm for this problem that achieves sublinear regret bounds under a fractional relaxation. We further introduce a randomized algorithm that is guaranteed to produce integer solutions with provably small installation cost. Simulation results show that our ROCR and OCR algorithms have better performance than other recent proposed policies and achieve a similar performance as the optimal static offline policy.

ACKNOWLEDGMENT

REFERENCES

[1] S. Kitanov, E. Monteiro, and T. Janevski, "5G and the fog — survey of related technologies and research directions," in *2016 18th MELECON*, pp. 1–6, 2016.

[2] M. T. Beck, M. Werner, S. Feld, and T. Schimper, "Mobile edge computing: A taxonomy," *Proc. of the Sixth International Conference on Advances in Future Internet.Citeseer*, 2014.

[3] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *IEEE INFOCOM*, pp. 235–243, 2019.

[4] G. S. Paschos, A. Destounis, and G. Iosifidis, "Online convex optimization for caching networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 625–638, 2020.

[5] X. Gao, X. Huang, Y. Tang, Z. Shao, and Y. Yang, "Proactive cache placement with bandit learning in fog-assisted IoT systems," in *ICC 2020*, pp. 1–6, 2020.

[6] T. Zhao, I.-H. Hou, S. Wang, and K. Chan, "Red/led: An asymptotically optimal and scalable online algorithm for service caching at the edge," *IEEE J-SAC*, vol. 36, no. 8, pp. 1857–1870, 2018.

[7] Y. Li, W. Dai, X. Gan, L. Fu, H. Ma, and X. Wang, "Cooperative service placement and scheduling in edge clouds: A deadline-driven approach," *IEEE Transactions on Mobile Computing*, 2021.

[8] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM*, pp. 207–215, 2018.

[9] R. Bhattacharjee, S. Banerjee, and A. Sinha, "Fundamental limits on the regret of online network-caching," *POMACS*, vol. 4, no. 2, pp. 1–31, 2020.

[10] S. Boyd and L. Vandenberghe, *Convex Optimization*. U.K. Cambridge Univ. Press, 2004.

[11] S. Shalev-Shwartz, "Online learning and online convex optimization.," *Foundations and Trends® in Machine Learning*, vol. 4, pp. 1935–8237, 2012.

[12] W. Wang and C. Lu, "Projection onto the capped simplex," *arXiv preprint arXiv:1503.01002*, 2015.

[13] S.-E. Elayoubi and J. Roberts, "Performance and cost effectiveness of caching in mobile access networks," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, pp. 79–88, 2015.

[14] J. L. Hellerstein, "Google cluster data. google research blog," Jan 2010 [Online]. https://github.com/google/cluster-data/blob/master/TraceVersion1.md.

[15] M. U. Thomas, "Queueing systems. volume 1: Theory (leonard kleinrock)," *SIAM Review*, vol. 18, no. 3, pp. 512–514, 1976.