Edge Service Caching with Delayed Hits: Minimizing Latency and Reducing Cost

Sikha Deka*, Radhika Sukapuram†
Dept. of CSE, Indian Institute of Information Technology Guwahati, India
Email: *sikha.deka@iiitg.ac.in, †radhika@iiitg.ac.in

***Abstract*—Multi-access Edge Computing (MEC) networks reduce service latency by caching services at the edge. A service request may be forwarded to the cloud for a response (Request Forwarding (R)), increasing Latency (L), and the service may be optionally downloaded, incurring a cost. If a download is in progress, the request may be buffered (a Delayed Hit (D)) if the remaining download time is less than the response time, adding to latency and previously overlooked. We argue that latency and cost must be considered separately, as the goal of MEC is to reduce latency. For the first time, we jointly consider Delayed Hits, Request Forwarding, resource constraints, download costs, and service caching, and introduce the DRL problem: minimizing Latency without assuming any request arrival pattern. We formulate this as an offline optimization problem (OPT). We propose an online algorithm called Online-DRL, derive its competitive ratio, and evaluate it using Google, SHARCNet and synthetic datasets. Compared to a simple baseline method, Online-DRL has up to 7.41% higher latency, but 92.54% lower cost, for Google traces. The results demonstrate that Online-DRL is a viable online algorithm for service caching.**

***Index Terms*—Service caching, Multi-access Edge Computing, Delayed hits, Latency, Online algorithms**

I. Introduction

In Multi-access Edge Computing (MEC), compute and storage are brought closer to users to reduce the latency and network congestion incurred while accessing the cloud. A *service* processes inputs from devices and generates outputs. A service for multiplayer Augmented Reality (AR) games that processes action data such as user gestures and provides a common response to all users [1], video data analytics engines that detect and respond to traffic accidents and hazards [2] etc. are examples. A service may be downloaded to the edge and cached to reduce latency. A service includes the code and the data associated with it, if any.

In MEC networks, when a request for a service arrives at an edge node, if the service is cached, the request is served from the cache. If the service is not cached, either: 1) the request is sent to the cloud (we call this *Request Forwarding*) so that it can be processed in the cloud, and the service is not downloaded. This may be because this service is not expected to be requested again, it is a service that has high fetch time and high resource requirements, etc. or 2) the request is sent to the cloud and when it is estimated that it will be required in the future, the service is downloaded and cached. When the service is being downloaded but is not yet available in cache, another service request that arrives may be buffered (we call this a *delayed hit*) or forwarded to the cloud, depending on whichever action results in lesser latency.

Delayed hits are relevant where the uplink bandwidth is significantly lower than the downlink bandwidth, and requests for the same service arrive in quick succession. With lower uplink bandwidth, the forwarding time of a service request increases. With high downlink bandwidth, the download time of a service decreases. Under these conditions, assume that a new request arrives quickly for a service whose download is already in progress. The time to receive a response if that request is forwarded to the cloud is likely to be longer than the time remaining to complete the download of that service, resulting in a delayed hit.

Recent work discusses the caching of services considering resource availability at the edge [3]. There are algorithms for service caching to minimize cost (defined as the sum of download and forwarding times) [4], and those that consider relaying (sending the service request to nearby caches) and request forwarding [5]. Fan et al. [6] have the objective of finding an online service caching algorithm with a provably small caching regret. *All the above discuss reducing or minimizing cost and do not consider delayed hits.*

A service request can be forwarded to the cloud and simultaneously a service download can be initiated. Since empirically the download cost is far higher than the cost of forwarding the service request [7], the *latency* in this case is due to forwarding the request to the cloud and getting a response, or waiting for the download to complete, if the time incurred for that is lower than the time to forward the request and get a response. The *cost* incurred is the cost to send a request to download a service to the cloud and then download the service. In content caching, in response to a request, the content must be downloaded *always* and there is no option to forward a request to the cloud to get a response. *Thus in service caching the total cost and the total latency are two different parameters* unlike in content caching [8]. *Since optimizing latency is an important requirement for low-latency edge services [9], we propose to minimize latency while keeping cost as a constraint.*

We address the following problem, for the first time, as far as we know: What is a deterministic online algorithm for caching and replacing services at the edge, without making any assumptions on the arrival patterns of requests, considering **D**elayed hits and **R**equest forwarding, and with the objective of minimizing ***L****atency* and reducing cost? We call this the DRL problem. Our contributions are: 1) We formulate the offline version of DRL as an optimization problem (OPT), analyze its complexity, and implement it. 2) We propose an

online algorithm for DRL called Online-DRL 3) We declare its competitive ratio to be $\gamma \in O(k(1+3\beta_{up}M_x))$ where M_x is the largest time to download a service and $\gamma \in \Omega(k(1+3\beta_{down}M_{min}))$, where M_{min} is the least time required to download a service. k is the maximum number of items in cache, and β_{up} and β_{down} the fraction of M_x and M_{min} for which requests are forwarded to the cloud and not buffered. 4) We evaluate Online-DRL using Google [10] and SHARCNet [11] cluster traces and compare its latency and cost with the offline optimal algorithm, OPT, and a simple version of the algorithm (LL-RC) that downloads a service when there is a cache miss, unless a download is in progress for the same service. Across all experiments for Google Cluster Traces, Online-DRL has a latency 0.016% (7.41%) higher than LL-RC in the best (worst) case, and the corresponding cost for Online-DRL is 77.57% (92.54%) lower than that of LL-RC. For SHARCNet, in all experiments, Online-DRL has 9.34% (26.62%) more latency than OPT in the best (worst) case, with the corresponding cost for Online-DRL being 87.30% (41.31%) higher than OPT. 5) We tradeoff cost and latency with a parameter called θ, for Online-DRL. 6) We also perform experiments using a synthetic dataset.

In our earlier work titled "Edge Service Caching with Delayed Hits and Request Forwarding to Reduce Latency" at the 2024 IFIP Networking Conference [12], we formulated OPT, proposed Online-DRL and evaluated the latter using a Google cluster dataset. The additions to that version are that 1) we have implemented OPT in Gurobi [13] and compared it experimentally with the online algorithm, 2) we have derived the competitive ratio of the online algorithm, 3) we have parameterized the online algorithm to tradeoff latency and cost and demonstrated its use experimentally, and 4) we have run experiments with the SHARCNet [11] dataset and a synthetic dataset for illustrating the algorithm better.

II. System Model and Optimization Formulation

An edge cache consists of the set of services that are running and stores the service id, the CPU, RAM and disk requirements of the service. C denotes the maximum number of services present in the cache. The variables used in the formulation are defined in Table I. The cost M_i is modeled as the time taken to download a service s_i. This consists of the time to send a request to the cloud to download the service and then to download the service from the cloud. The objective is to minimize the total latency of all service requests (Eq.1). The status of a service in cache is indicated as 0 (s_i is in cache after its j_ith request) or 1 (Eq. 2). The decision to download a service is indicated as 0 (the download is not initiated) or 1 (Eq. 3). The total CPU (RAM,disk) requirement of the cached items must be less than or equal to C_{cpu} (Eq. 5)(C_{ram},(Eq. 6), C_{disk}, (Eq. 4)). The total number of items cached must be less than C (Eq.7). For the 1st request of any service s_i, there is no item s_i in the cache (Eq. 8).

j_i indicates an instance of a request for service s_i. A service can be downloaded, evicted and then again downloaded. Since a service may be downloaded more than once, z'_i contains all the instances (the values of j_i) of s_i at which download occurs

TABLE I: List of symbols used

Variable	Description
s_i	Service i
$d(s_i, j_i)$	Latency of the j_ith request of service s_i
N	The total number of requests
$x(s_i, j_i)$	0 indicates that s_i is in the cache after the j_ith request for s_i and 1 otherwise
$r(s_i, t)$	The number of requests for s_i until the tth request (across requests)
S_i	The size of s_i
C_{disk}	The maximum disk capacity of the edge
CPU_i	The CPU requirement of s_i
C_{cpu}	The maximum CPU capacity of the edge
RAM_i	RAM of s_i
C_{ram}	The maximum RAM capacity of the edge
C	The maximum number of items that can be cached
M_i	The time taken to download s_i
$y(s_i, j_i)$	1 indicates that download is initiated or in progress at the j_ith request for s_i and 0 indicates otherwise
z'_i	The set of instances of s_i at which download begins
z_i	An element of z'_i
z_m	The earliest instance of s_i at which download is completed
l_i	The time taken for the service request for s_i to send a request to the cloud and get a response.
M_{max}	The maximum cost
$T(x)$	The time stamp of the xth request
f_p	The pth service request across services
$A(s_i, f_p)$	Function to find the value of j_i corresponding to f_p
U	The maximum number of unique requests

(Eq. 11). The optimization algorithm must choose a value of z_i that is less than or equal to j_i and is the maximum value in z'_i that is less than or equal to j_i(Eq. 9).

$y(s_i, j_i)$ is set to 1 to indicate that the download is initiated or in progress at the j_ith request for s_i and 0 indicates otherwise. When download is initiated at $j_i = z_i$, all of $y(s_i, z_i), y(s_i, z_i + 1), \ldots, y(s_i, z_m)$ are equal to 1 (Eq.12). z_m is the earliest instance of s_i at which the download is completed. After this (and also before the download is initiated), $y(s_i, z_i)$ continues to be 0 (Eq.12). Let f_p be the pth service request (across services) (Eq. 13). The condition $(T(f_p) \geq (T(z_i) + M_i))$ is checked, where M_i is the download cost for s_i, to know if the download is complete, and after that the service s_i is cached (Eq. 14). The condition $(T(f_p - 1) < (T(z_i) + M_i))$ is checked in Eq. 14 to ensure that for the first request of any service after download is completed, the service is cached, and not earlier. After it is cached once, it may or may not remain in the cache.

If download does not begin at the j_ith request and s_i is not cached, then the latency is l_i, as the service request is then forwarded to the cloud. If s_i is found cached at the j_ith request, then the latency is 0 and if so, the service is not downloaded. Downloading a service that is already cached is not allowed and therefore, the delay incurred in this case is represented as ∞. If download of s_i began at z_i, then at $T(z_i) + M_i$ the service will be downloaded. At j_i, the remaining time before the service is downloaded is $T(z_i) + M_i - T(j_i)$. The least of l_i and that value is the latency of the j_ith request. The above are represented by Eq. 15. The total cost to download must be less than the maximum cost M_{max} (Eq. 16). The optimizer can choose to set $x(s_i, j_i)$ to 1 and evict a service whenever required.

$$min \ \Sigma_{i=1}^{U} \ \Sigma_{t=1}^{N} \ d(s_i, r(s_i, t)) \tag{1}$$

$$x(s_i, j_i) \in \{0, 1\} \quad \forall i \tag{2}$$

$$y(s_i, j_i) \in \{0, 1\} \quad \forall i \tag{3}$$

$$\Sigma_{i=1}^{N} S_i(1 - x(s_i, r(s_i, t))) \leq C_{disk} \quad \forall t \tag{4}$$

$$\Sigma_{i=1}^{U} CPU_i(1 - x(s_i, r(s_i, t))) \leq C_{cpu} \quad \forall t \tag{5}$$

$$\Sigma_{i=1}^{U} RAM_i(1 - x(s_i, r(s_i, t))) \leq C_{ram} \quad \forall t \tag{6}$$

$$\Sigma_{i=1}^{U}(1 - x(s_i, r(s_i, t))) \leq C \quad \forall t \tag{7}$$

$$x(s_i, 1) = 1 \quad \forall i \tag{8}$$

$$z_i \leq j_i \ \mid z_i \text{ has the maximum value possible } \forall i \tag{9}$$

$$z_i, j_i \in \mathbb{Z} \ \forall i \tag{10}$$

$$z_i \in z_i' \tag{11}$$

$$y(s_i, b_i) = \begin{cases} 1, \\ \quad \text{if } \ b_i \in \{z_i, z_i + 1, \ldots, z_m\} \text{ where } T(z_i) \leq \\ T(z_m) \leq (T(z_i) + M_i), \ \forall i \\ 0, \\ \text{otherwise} \end{cases} \tag{12}$$

$$f_p \in \mathbb{Z} \tag{13}$$

$$x(s_i, A(s_i, f_p)) = 0, \ \forall i \text{ if } (T(f_p) \geq (T(z_i) + M_i)) \wedge (T(f_p - 1) < (T(z_i) + M_i)) \tag{14}$$

$$d(s_i, j_i) = \begin{cases} min(l_i, T(z_i) + M_i - T(j_i)) \\ \quad \text{if } x(s_i, j_i) = 1 \wedge y(s_i, j_i) = 1, \forall \ i, j_i \\ l_i \\ \quad \text{if } x(s_i, j_i) = 1 \wedge y(s_i, j_i) = 0 \quad \forall i, j_i \\ 0 \\ \quad \text{if } \quad x(s_i, j_i) = 0 \wedge y(s_i, j_i) = 0 \quad \forall i, j_i \\ \infty \\ \quad \text{if } \quad x(s_i, j_i) = 0 \wedge y(s_i, j_i) = 1 \quad \forall i, j_i \end{cases} \tag{15}$$

$$\Sigma_{i=1}^{U} y(s_i, z_i) * M_i < M_{max} \tag{16}$$

The formulation is a Mixed Integer Quadratic Programming problem when implemented in Gurobi [13] (called OPT) and the problem is therefore in NP [14]. OPT assumes that all service requests are known in advance and cannot be used in a real network. Since service requests continue to arrive at the edge and need to be processed online, we propose an online algorithm called "Online-DRL", described below.

III. Online algorithm for DRL

The main procedure: The main procedure of Algorithm 1, Online-DRL, (line 1) runs in an infinite loop, processing services one by one, as they arrive. It checks if any services that are in the pending list to be downloaded have completed downloading. If so, some items are evicted from the cache and the downloaded item is cached (line 6). If a request for a service arrives, it is processed (line 9). HANDLE_SERVICE_REQUEST() (line 9) must decide when to download a service if it is not cached. After downloading starts, it must decide whether subsequent requests for the same service must be buffered or forwarded to the cloud.

Handling each service request: If a download for a service s_i is initiated the first time its request r_i arrives at the edge at time t_i, the download would complete at time $t_i + M_i$ (M_i is the time taken to download s_i). Then s_i would be cached. Therefore, a request for s_i at $t_i + M_i$ units and later will not be missed. T_i is the time from the first request for s_i, or the time from the first request for s_i since it was evicted from cache (tracked by $miss_i$ in line 12 in Algorithm 1). If T_i is greater than or equal to M_i (line 15), s_i is downloaded.

If a download of s_i has not yet been initiated and another request for s_i arrives before $t_i + M_i$ units of time, it will incur a latency of l_i. This latency is due to the request being forwarded to the cloud. Thus, if L_i, the sum of the latencies of requests for s_i so far is greater than or equal to M_i, then also s_i is downloaded (line 15). T_i and L_i are calculated by the procedure GETPENALTY() in Algorithm 1. Both retrospective downloads for s_i assume that there will be additional requests for s_i.

If a request arrives while the service requested for (s_i) is being downloaded and if the remaining time to complete the download is less than l_i, the service request is buffered (line 23), otherwise it is forwarded to the cloud (line 25). If the request is a hit, its credit is set to M_i (line 27). If s_i is to be cached, its cost is set to M_i (line 6).

Algorithm for eviction from cache: Algorithm 2 is called after an item is downloaded and must be cached (line 6) and when there is a cache hit (line 27). Now we describe Algorithm 2, which is a modified version of an algorithm for content caching [15]. When a service is downloaded, if any of the CPU, RAM and disk requirements for the service is going to exceed the respective maximum permissible limits, or if the number of items exceeds the maximum allowed, items need to be evicted from cache (line 4 of Algorithm 2). Each item has its credit initialized to its M_i when it is cached. Credits are recalculated each time an item needs to be evicted from the cache. After reducing the credit for each item (line 26), the set of files with credit 0 is eligible for eviction. In addition, from each of the items in this list, the resource requirements of s_i that have not been met so far are subtracted (line 9). If the value is negative, it is set to 0. For each item in this list, the CPU, RAM and disk values are added (line 10). Now, the items are sorted in ascending order by the sum and stored in $sorted\_f$ (line 10). Each item in $sorted\_f$ is evicted until the resource requirements of g are met. If the resource requirements are not met even after evicting all the items in $sorted\_f$, the procedure repeats (line 6). If the number of items in the cache exceeds C (line 17), the credits of the items are calculated, and one item whose credit is 0 is evicted. If g is already cached, its credit is reset to $cost$. For ease of exposition, we assume that no item exceeds the maximum limits of CPU, RAM and disk.

Competitive ratio: Let $\mathcal{U}$ be the set of all possible requests. Consider some input $I \in \mathcal{U}$. An online algorithm ALG has a competitive ratio of γ if $\max\left\{\frac{ALG(I)}{OPT(I)}\right\} = \gamma$. $ALG(I)$ is the value of the solution produced by the online algorithm ALG and $OPT(I)$ is the value of the solution produced by the optimal offline algorithm.

Theorem 1. *The competitive ratio of Online-DRL is γ where $\gamma \in \Omega(k(1 + 3\beta_{down})M_{min})$ and $\gamma \in O(k(1 + 3\beta_{up})M_x))$*

Algorithm 1 Online algorithm to reduce latency: Online-DRL

procedure MAIN
  ▷ r_i is a service request, s_i is a service corresponding to it and l_i the latency incurred in forwarding the request to the cloud and getting a response. M_i is the download cost of service s_i. $pending = NULL$. $miss_i = -1$ when a service is requested first or when a service is evicted from cache. cpu, ram and $disk$ are the values corresponding to CPU, RAM and disk requirements of the current request.
  while True **do**
    for each $s_i \in pending$ **do**
      if s_i has completed download **then**
        LANDLORD-RR(s_i,M_i,cpu,ram,$disk$)
        Remove s_i from $pending$
        $miss_i = -1$
    if a request for a service arrives **then** HANDLE_SERVICE_REQUEST(s_i,l_i,r_i,$tstamp$,$miss_i$)
procedure HANDLE_SERVICE_REQUEST(s_i, l_i, r_i, $tstamp$, $miss_i$)
  if s_i is not cached **then**
    If $miss_i$ is -1, $miss_i = tstamp$ ▷ Time stamp of the very first miss or the first miss after an eviction
    (T_i, L_i) =GETPENALTY(r_i,$tstamp$,l_i,$miss_i$)
    if ($T_i \geq M_i$ or $L_i \geq M_i$) and $s_i \notin pending$ **then**
      Initiate download
      Add s_i to $pending$
      Forward r_i to cloud ▷ A miss
    else if ($T_i < M_i$ or $L_i < M_i$) and $s_i \notin pending$ **then**
      Forward r_i to cloud ▷ A miss
    else
      if $s_i \in pending$ **then**
        if remaining time to download $\leq l_i$ **then**
          Buffer r_i ▷ A delayed hit
        else
          Forward r_i to cloud ▷ A miss
  else
    LANDLORD-RR(s_i,M_i,cpu,ram,$disk$) ▷ A hit
procedure GETPENALTY(r_i,$tstamp$,l_i,$miss_i$)
  $T_i = tstamp - miss_i$
  $L_i = l_i$∗Number of request forwards to the cloud from the time $miss_i$, including $miss_i$
  return (T_i, L_i)

where M_x is the largest time to download a service, M_{min} the smallest time to download a service, k the maximum number of entries in cache, and β_{up} (β_{down}) the fraction of M_x (M_{min}) requests that are forwarded to the cloud because the latency to forward is less than the remaining time to download.

Proof. Time is divided into discrete steps. Without loss of generality, we assume that M_i is a positive integer that represents the number of steps required to download a service s_i from the cloud. M_i also corresponds to the maximum number of service requests that can arrive in the system while a service is being downloaded. At step M_i, the service completes the download, thus taking $M_i - 1$ time steps to download; it also entails a maximum of M_i requests. If there is a series of M_i requests and the download begins for the first request, the M_ith request can be served from cache. The first half of a time step is to accept the service request and act on it (forward it, download the service), and the second half is to accept downloads if any and act on them (evict a service from cache if full).

Consider the sequence of service requests where a service request s_i is followed by M_i time slots where there is no request, which is followed by βM_i time slots with s_i requested in each of the βM_i slots, followed by no requests in the next M_i slots. β is the fraction of M_i requests for which it incurs less latency to forward the request to the cloud than wait for the download to complete (no delayed hits). This is denoted as $s_i 0^{M_i} s_i^{\beta M_i} 0^{M_i}$. The above sequence is called a *Basic Sequence (BS)*.

Consider BSs called BS1, BS2 and BS3, where each of these BSs contain $k/3$ *different* services . $BS1$ is $s_i 0^{M_l} s_i^{\beta_l M_l} 0^{M_l}$ repeated $k/3$ times, $BS2$ and $BS3$ are $s_i 0^{M} s_i^{\beta M} 0^{M}$ repeated $k/3$ times. Each service in BS1, BS2 and BS3 have resource requirements as $(M_l, \mathcal{R}, \mathcal{C})$, $(M, \mathcal{R}_l, \mathcal{C})$ and $(M, \mathcal{R}, \mathcal{C}_l)$ where each tuple denotes the download time, RAM and CPU requirements, respectively. The subscript l here indicates a large requirement for that resource, and if there is no subscript, it indicates negligible requirements for that resource. Furthermore, $B*(k/3)(M_l+M+M) = C_{disk}$, $(k/3)(\mathcal{R}_l+\mathcal{R}+\mathcal{R}) = C_{ram}$ and $(k/3)(\mathcal{C}_l + \mathcal{C} + \mathcal{C}) = C_{cpu}$, where B is the downlink bandwidth. That is, the services in these three BSs can be downloaded without exceeding any of the resource requirements, and the number of items in the cache and downloading them fills up the cache.

A sequence denoted as $s_i 0^{M_m} s_i 0^{M_m}$ is called a *Mini Sequence (MS)*. Let $M_m = (k/3)M_l$ denote the time to download this service. $\mathcal{R}_m = (k/3)R_l$ denotes its RAM requirement and $\mathcal{C}_m = (k/3)C_l$ denotes its CPU requirement. A *Large Sequence* consists of the following sequence: BS1, BS2, BS3, MS. The Large Sequence repeats n times. The intuition is that the service in MS, if downloaded, would require evicting all the items in the cache. This is because even if the services corresponding to $(k/3)M_l$ are evicted, since the RAM and CPU values corresponding to these are smaller than R_m and C_m, respectively, the service whose download time is M_m cannot be cached. On the other hand, there will not be enough disk space if the $k/3$ services corresponding to R_l (or C_l) are evicted[1]. OPT will not do this, as it can foresee the future and forward these requests always.

When the first request of BS1 arrives, OPT forwards it to the cloud, incurring a latency of l and downloads and caches the service. Subsequent requests of the BS therefore incur 0 latency. The latency incurred by OPT for BS1 is l. Since there are $k/3$ of them, the latency incurred is $l * k/3$, which is also the case for BS2 and BS3. For MS, since there are only

[1]For example, consider $k = 6$, $B = 1$, $M_l = R_l = C_l = 5$, $M = R = C = 1$, $M_m = R_m = C_m = 10$. Let the maximum limits of disk, RAM and CPU be 10 each. To cache M_m, $k/3 = 2$ resources of disk size 5 will need to be evicted. However, they correspond only to RAM and CPU sizes of $1 + 1 = 2$ each. Therefore, the entire cache must be evicted.

two requests and the current cache will have to be completely evicted to store this service, it instead forwards the requests and incurs a latency of $2l$ every time it receives MS. Thus, the total latency for LS repeated n times is

$$OPT_t = 3l(k/3) + 2ln = (k + 2n)l \tag{17}$$

In the case of Online-DRL, the first request of the BS incurs a latency of l, and the second request after M_l units of time triggers a download. For each of the subsequent $\beta_l M_l$ requests, it forwards the requests to the cloud, by definition. Thus, the latency for Online-DRL for the first BS1 is $l + \beta_l M_l l$. There are $k/3$ of them, with a total latency of $(k/3)(l+\beta_l M_l l)$. BS1 and BS2 result in latencies of $(k/3)(l+\beta M l)$ each. When the first request of MS arrives, Online-DRL forwards it. It also forwards the next request of MS and downloads the service, incurring a total latency of $2l$ for MS. This repeats for all n LSs. Thus, the total latency due to n LSs is

$$\begin{aligned} DRL_b =& n[(k/3)(l + \beta_l M_l l) + 2(k/3)(l + \beta M l) + 2l] \\ =& ln[(k/3)(1 + \beta_l M_l) + 2 * (k/3)(1 + \beta M) + 2] \end{aligned} \tag{18}$$

From Eq. 17 and Eq. 18, the competitive ratio is

$$\gamma = \frac{ln[(k/3)(1 + \beta_l M_l) + 2(k/3)(1 + \beta M) + 2]}{l(k + 2n)} \tag{19}$$

Assuming $n > k$ and substituting n for k in the denominator,

$$\begin{aligned} \gamma &> \frac{ln[(k/3)(1 + \beta_l M_l) + 2(k/3)(1 + \beta M) + 2]}{3nl} \\ &> (k/9)(3 + \beta_l M_l + 2\beta M) + 2/3 \end{aligned} \tag{20}$$

Thus, $\gamma \in \Omega(k(1 + \beta_l M_l + 2\beta M))$, which can be simplified to $\gamma \in \Omega(k(1 + 3\beta_{down} M_{min})$, where M_{min} is the least time required to download a service. $\beta_{down} < \beta_l$ and $\beta_{down} < \beta$.

If the numerator and denominator of Eq. 19 are divided by n, as n becomes very large, $\gamma \in O(k(1 + \beta_l M_l + 2\beta M))$. This can be simplified to $\gamma \in O(k(1 + 3\beta_{up} M_x))$ where M_x is the largest time to download a service. It is always possible to find $\beta_{up} > \beta_l$ and $\beta_{up} > \beta$, as, for a larger fraction of M_x, requests would continue to be forwarded to the cloud.

The cost incurred by OPT is $(k/3)(M_l + 2M)$, which is in $O(k(M_l + M)) = O(kM_x)$ and that by Online-DRL is $((k/3)M_l + (2k/3)M + M_m)n$, which is in $O(nk(M_l + M) + nM_m)) = O(knM_x)$. The latter is the upper limit for cost. □

Trading off cost for latency: We have parameterized Online-DRL by defining the download condition as: $T_i \geq \theta \cdot M_i$ or $L_i \geq \theta \cdot M_i$. θ is a tunable parameter that can take any real value greater than or equal to 0. When $\theta = 0$, downloads occur frequently, resulting in higher costs, and as θ increases, downloads are less frequent, leading to lower costs. Thus, in Algorithm 1 line 14 will change to "If ($T_i \geq \theta \cdot M_i$ or $L_i \geq \theta \cdot M_i$) and $s_i \notin pending$" and line 18 will change to "ElsIf($T_i < \theta \cdot M_i$ or $L_i < \theta \cdot M_i$) and $s_i \notin pending$".

Algorithm 2 Landlord With Resource Checks

Each service g has a real value $credit[f]$ and CPU, RAM and Disk resources associated with it. The maximum number of items in cache is C. Each of CPU, RAM and Disk resources has a maximum permissible value.
procedure LANDLORD-RR(g, $cost$, cpu, ram, $disk$)
  if g is not in the cache **then**
    if adding g to cache exceeds the maximum resource requirements of CPU, RAM or Disk **then**
      $c = 0$, $r = 0$, $d = 0$ ▷ To store the cumulative requirements
      while True **do** ▷ Keep evicting files until the resource requirements of g are met
        $eligible\_for\_eviction$ = DECREASECREDIT()
        for each item f in $eligible\_for\_eviction$ **do**
          Subtract $max(0, cpu–c)$,$max(0, ram−r)$, $max(0, disk - d)$ from those of f. If a value is negative, store it as 0.
        Sort the modified $eligible\_for\_eviction$ in ascending order of the sum of cpu,ram and $disk$ and store in $sorted\_f$ ▷ As less items as possible are evicted to accommodate the new entry
        for each item f in $sorted\_f$ **do**
          Evict f
          Add the cpu, ram and $disk$ values of f to c, r and d respectively ▷ Update the cumulative requirements
          if all resource requirements of g are met now **then**
            Cache g, setting its $credit = cost$ and CPU, RAM and Disk values to cpu, ram and $disk$
            return
    if number of items in cache exceeds C **then**
      $eligible\_for\_eviction$ = DECREASECREDIT()
      Evict an item from $eligible\_for\_eviction$
      Cache g, setting its $credit = cost$ and CPU, RAM and Disk values to cpu, ram and $disk$
      return ▷ Only one item needs to be evicted
    else
      Cache g, setting its $credit = cost$ and CPU, RAM and Disk values to cpu, ram and $disk$
  else ▷ g is in the cache
    Reset $credit[g]$ to cost(g).
procedure DECREASECREDIT()
  For each item f in cache, decrease $credit[f]$ by $\Delta \cdot$ size$[f]$, where $\Delta = \min_{f \in \text{cache}} \frac{credit[f]}{size[f]}$.
  $items$ = Items from $cache$ with $credit[f] = 0$
  return $items$

IV. Implementation and Evaluation

Datasets: We conduct experiments using cluster traces from Google [10] and from SHARCNet [11] and a synthetic dataset. Google Cluster Traces comprise Job and Task events files which are processed to create datasets. Each unique JobID occurrence with distinct timestamps represents a service request and we consider only records with event type "SUBMIT". Eleven preprocessed output files named as "dataset0", "dataset1", ..., "dataset10" are generated, each including JobID, Timestamp, CPU, RAM, Disk, Task Index,

TABLE II: Parameters and Their Default Values

Parameter	Description	Google	SHARCNet	Synthetic
T_{max}	Maximum download cost (Only applicable for OPT)	Not used	Total download cost from Online-DRL	Not used
l_{size}	Size of a forwarding request (or response)	$1/10$ of smallest disk size	$1/10$ of smallest disk size	$1/10$ of smallest disk size
$sizefactor$	multiplication factor for disk size	2^{30}	1	2^{10}
U	Uplink bandwidth	240 Mbits/s	819200 Mbits/s	240 Mbits/s
D	Downlink bandwidth	320 Mbits/s	2457600 Mbits/s	320 Mbits/s
C	Number of items in cache	50	5	5
T	Time horizon (size of the dataset)	41349	1000	50000
C_{cpu}	Maximum edge CPU capacity	∞	∞	∞
C_{ram}	Maximum edge RAM capacity	∞	∞	∞
C_{disk}	Maximum edge disk capacity	∞	∞	∞

and Status, with values equal to 0 in CPU, RAM, or Disk fields replaced by the median value. The disk size is assumed to be in KB. We use dataset2, unless otherwise stated.

From the original SHARCNet dataset, we selected the following columns of interest: ExecutableID, Submit Time, UsedCPUTime, UsedMemory, JobID, and Status. Since all the values in the original file for UsedLocalDiskSpace were -1, we copied the values from the UsedMemory column into the UsedLocalDiskSpace column. UsedCPUTime is the calculated average CPU time used across all processors. UsedMemory and UsedLocalDiskSpace are the average memory usage per processor. To preprocess the Submit Time data, we subtracted the first Submit Time value from each value in the Submit Time column for all rows as the original timestamps were large. We incremented the Submit Time by 1 wherever the values were identical. Entries with values 0 for CPU, RAM and disk were replaced with the median value of the column with that entry. If the CPU, RAM, and Disk values of an executable vary across occurrences, we use the value in its first occurrence. Submit Time and CPU time are measured in seconds, while the RAM and Disk values are in MegaBytes. The RAM and Disk values are compared against their corresponding MAXLIMIT values, also expressed in MegaBytes.

We also generate a synthetic dataset by taking the unique services in Google dataset2, then creating a large dataset with the services following a Zipf distribution, and arrival times following a Poisson distribution [16].

Evaluation: We compare Online-DRL with an algorithm that immediately downloads a service when requested if the service is not cached or if its download is already not in progress. This calls Algorithm 2 to cache the downloaded service. We call this algorithm LL-RC.

The parameters for all experiments and their default values are as shown in Table II. The following metrics are evaluated for a given time horizon T: a) the total latency of all requests b) the total cost c) Number of hits and delayed hits.

The total latency due to OPT, denoted as OPT_t is as per Eq. 17. The total latency, the total cost, and the number of hits and delayed hits resulting from Online-DRL are compared with the optimal latency (OPT), the minimum total theoretical optimal latency (OPT_t), and $LL-RC$.

C refers to the number of distinct service items that the cache can hold, irrespective of the resource requirements of each item. Repeated requests for the same service has the same resource requirements. The OPT program is run exclusively for the SHARCNet dataset. Due to its high computational time, the calculation is limited to only 1000 rows. The SHARCNet dataset has 420 unique services in 1000 rows and they are interspersed. In the Google dataset, there are fewer unique services that can be cached. In the Google dataset only 30 unique services are found in 1000 rows, and they are mostly grouped together. This makes the caching constraint C less meaningful, as interspersed services are needed for C to matter. Therefore, we did not run the OPT program for the Google dataset.

Since the algorithms are deterministic, each experiment is run only once. The Online-DRL and $LL-RC$ experiments were conducted on a system with an Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz and 8 GB RAM. The optimization formulation, OPT, was implemented using the Gurobi Optimizer (version 11.0.3, build v11.0.3rc0) on a Linux system with Ubuntu 20.04 and was run on a 16-core Intel(R) server with the following specifications: Xeon(R) Silver 4208 CPU@2.10GHz with 64 GiB RAM. A broken y-axis is used in Fig. 1b, 2b, 3b, 4b, 5b, 10b and 11b to improve readability.

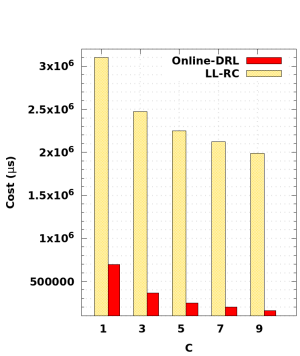
In all of the experiments, we observe that the cost of Online-DRL is significantly lower than that of LL-RC, while LL-RC exhibits slightly better latency. *This is because LL-RC downloads every service that is not cached, unlike Online-DRL, which downloads conditionally.* The latency of OPT_t increases with n and k, as per Eq.17.

A. Experiments using the Google dataset:

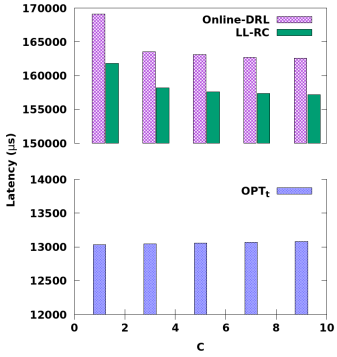
Experiment 1: The results of *Experiment 1*, studying the impact of varying C for a given T are shown in Fig. 1a, Fig. 1b and Fig. 1c. The latency of both Online-DRL and LL-RC decreases as the value of C increases. Delayed Hits are not calculated for LL-RC as it always downloads. In Figs. 1a and 1b, although the latency of Online-DRL is slightly higher than that of LL-RC, the cost of LL-RC is much higher, as it downloads every service that is not cached. From Fig. 1c, it is clear that LL-RC achieves a higher number of hits, but at a high cost.

Experiment 2: In *Experiment 2*, the first 3 Google datasets are concatenated to create a new dataset A. We study the impact of varying T using *the same* data set A assuming the default value of C. From A the first 25000 ($DS1$), 50000 ($DS2$), 75000 ($DS3$) and 100000 ($DS4$) lines are taken, starting from the first entry each time. The results are in Fig. 2a, Fig. 2b, and Fig. 2c. It is observed that using Online-DRL results in higher latency comparable to LL-RC, with much lower cost than LL-RC.

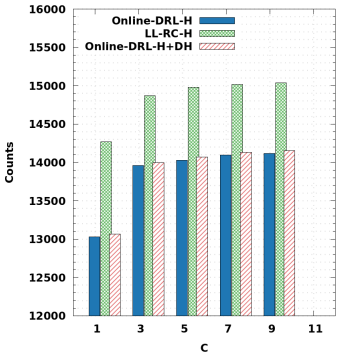
Experiment 3: *Experiment 3* examines the impact of varying uplink bandwidth (shown in Fig. 3a, Fig. 3b, and Fig. 3c).



(a) Cost for various cache sizes

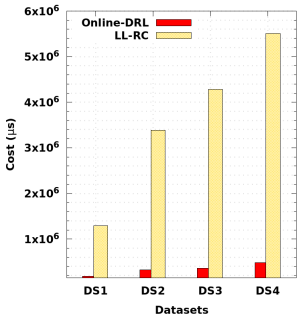


(b) Latency for various cache sizes

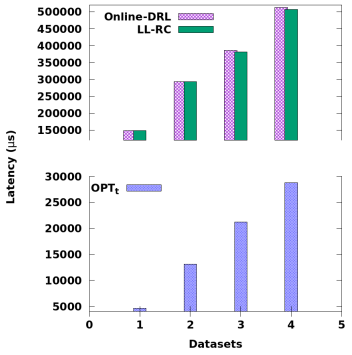


(c) Hits, Delayed hits for various cache sizes

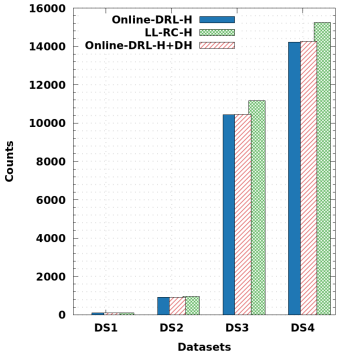
Fig. 1: Exp. 1: Online-DRL and LL-RC for a smaller time horizon, varying the cache size C.



(a) Cost for increasing T

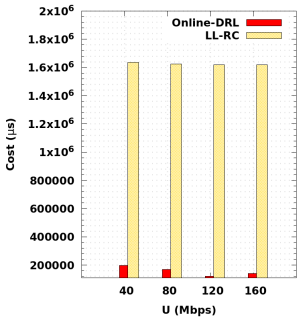


(b) Latency for increasing T

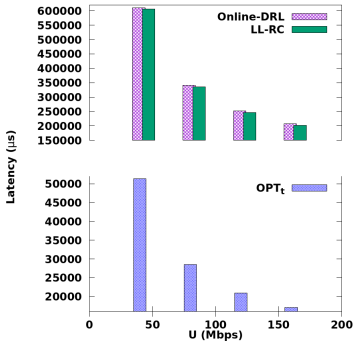


(c) Hits, Delayed hits for increasing T

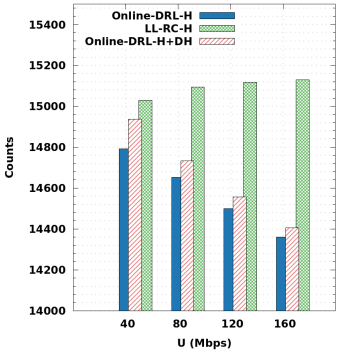
Fig. 2: Exp. 2: Online-DRL and LL-RC for different time horizons, for C=50.



(a) Cost for various uplink speeds



(b) Latency for various uplink speeds



(c) Hits, Delayed hits for various uplink speeds

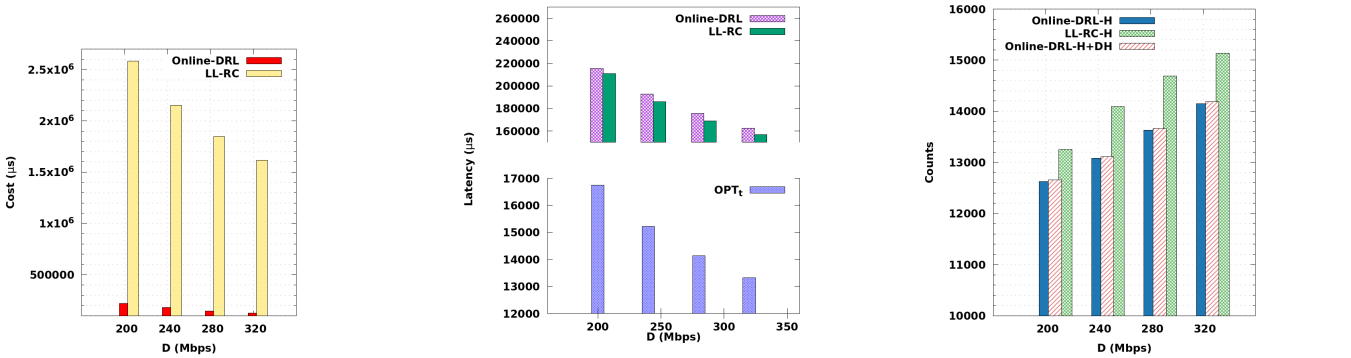
Fig. 3: Exp. 3: Online-DRL and LL-RC with various uplink speeds.

As seen in Fig. 3b, there is a significant drop in latency for both Online-DRL and LL-RC as the uplink bandwidth increases from 40 to 80. This is because requests can be forwarded faster to the cloud. From Fig. 3c, as the uplink bandwidth increases, the hits are marginally higher for LL-RC, as it always downloads services. Hits and delayed hits for Online-DRL decrease with increasing uplink bandwidth, as lower values of L_i result in fewer downloads. When U increases, requests are forwarded faster. Therefore, for the same number of requests, L_i reduces, lowering the number of downloads and the number of hits. However, latency decreases as it takes less time to forward requests to the cloud.

Experiment 4: The results of *Experiment 4*, which examines the impact of varying downlink bandwidth, are shown in Fig. 4a, Fig. 4b, and Fig. 4c. Varying downlink bandwidth has a small impact on latency when comparing Online-DRL and LL-RC. The response to a forwarded request will arrive faster at the edge due to increased downlink bandwidth. Since the response size is small, the improvement in speed is minimal. However, Fig. 4a shows a significant difference in cost. This is because the downlink bandwidth directly affects the time to download, that is, the cost.

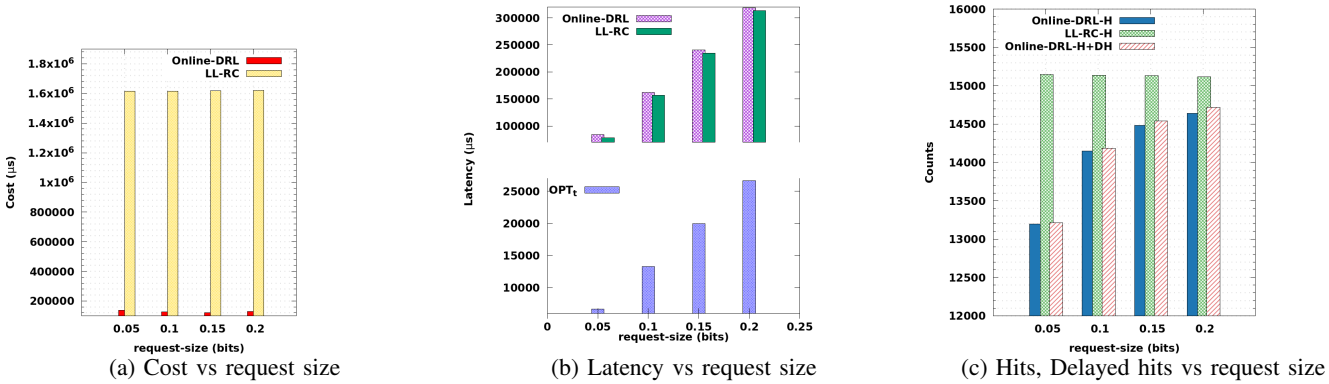
Experiment 5: *Experiment 5* studies the impact of varying request sizes (Fig. 5a, Fig. 5b, and Fig. 5c). The latency increases as the request sizes increase, as shown in Fig. 5b. This is because it takes more time to forward each request to the cloud. Since the downlink bandwidth remains the same, the impact on cost is not significant.

Experiment 6: The results of *Experiment 6*, studying the impact of varying the maximum limits of C_{disk}, C_{cpu} and C_{ram} are in Fig. 6a, Fig. 6b and Fig. 6c. C is set to 50000, which is larger than the number of service requests, to study the effect of varying the maximum limits for CPU, RAM and Disk size. Therefore, the theoretical latency value is not



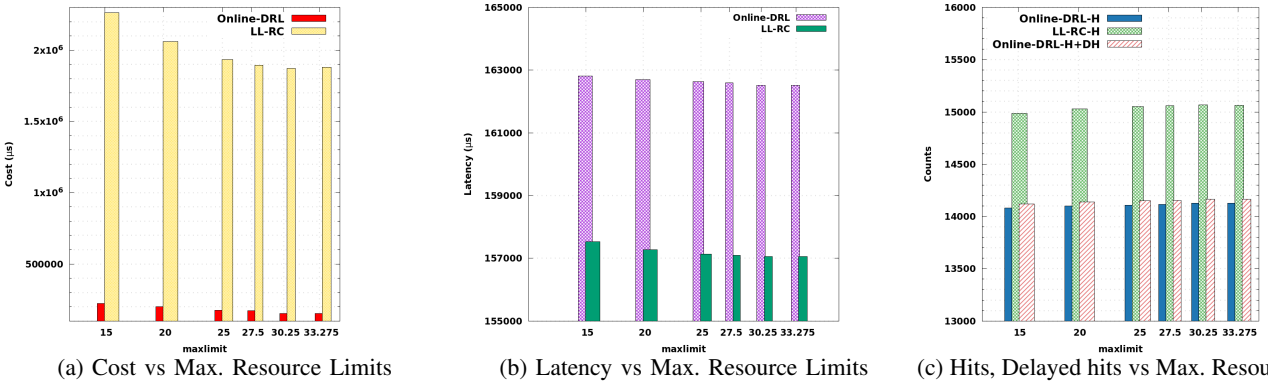
(a) Cost vs downlink speed (b) Latency vs downlink speed (c) Hits, Delayed hits vs downlink speeds

Fig. 4: Exp. 4: Online-DRL and LL-RC for various downlink speeds.



(a) Cost vs request size (b) Latency vs request size (c) Hits, Delayed hits vs request size

Fig. 5: Exp. 5: Online-DRL and LL-RC for various request sizes



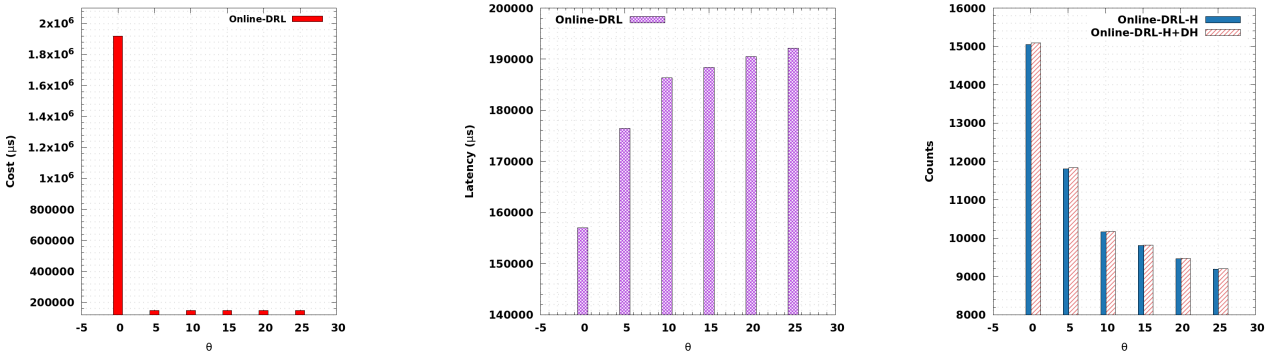
(a) Cost vs Max. Resource Limits (b) Latency vs Max. Resource Limits (c) Hits, Delayed hits vs Max. Resource Limits

Fig. 6: Exp. 6: Online-DRL and LL-RC for various maximum resource limits.

meaningful and is not calculated here. The value of each resource limit is set to the maximum of the following: a) the product of x (the value on the x-axis) and the median value of the resource, and b) the maximum resource requirement of any item in the dataset. As the maximum resource limits increase, more services can then be cached, causing latency to decrease.

Experiment 7: In this experiment, θ is varied and the cost and latency of Online-DRL are observed. Here, the cache size $C = 10$. The cost of Online-DRL, as shown in Fig. 7a, is significantly high when $\theta = 0$. This is because for $\theta = 0$, Online-DRL is the same as LL-RC as far as the download initiation condition is concerned. That is, there is a download for every service request. Note that LL-RC is not included in this experiment since it always downloads services. A trade-off between decreasing cost (Fig. 7a) and increasing latency (Fig. 7b) can be observed. As θ increases, the number of downloads decreases, leading to a corresponding reduction in the number of hits, as observed in Fig. 7c. Furthermore, delayed hits are observed to be low in Fig. 7c.

Experiment 8: We vary the uplink bandwidths and subsequently the downlink bandwidths and measure the number of delayed hits (Fig. 8). An increase in uplink bandwidth reduces the time required to forward a request, making forwarding more efficient than buffering the request while waiting for the download to complete. This leads to a decrease in the number of delayed hits. In contrast, as the downlink bandwidth increases, buffering becomes comparatively more effective: the latency introduced by waiting for the download to complete is reduced, making buffering preferable to forwarding under



(a) Cost vs θ (b) Latency vs θ (c) Hits, Delayed hits vs θ

Fig. 7: Exp. 7: Online-DRL for varying θ.

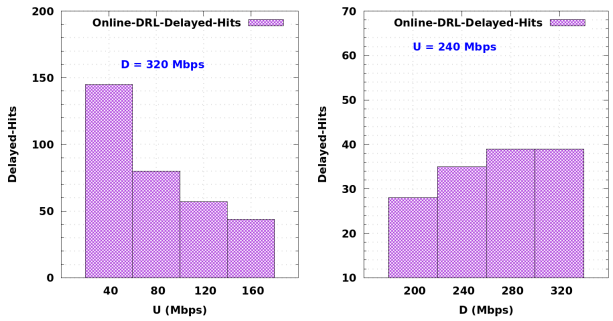


Fig. 8: Exp. 8: Delayed hits for varying U and D

high downlink bandwidth conditions.

B. Experiments using the SHARCNet dataset:

Larger values are considered for the uplink and downlink bandwidths in the SHARCNet experiments because the disk sizes are larger there. In each experiment, the cost of Online-DRL is first measured. The maximum cost of OPT, that is, T_{max}, is then set to this value. The cost incurred by OPT does not vary smoothly as it can find solutions with costs below T_{max}, but the proximity of these costs to T_{max} can vary. OPT always performs better in terms of cost and latency, as it is the optimal algorithm.

Experiment 9: Fig. 9 compares the performance of Online-DRL, LL-RC, and OPT while varying the cache size for a time horizon of 1000 service requests (1000 rows). As shown in Fig. 9a, the cost fluctuates between different cache sizes for OPT. OPT performs the best and LL-RC the worst in terms of cost, as the number of downloads is very high for the latter. In some cases in Fig. 9b, LL-RC has a better latency than OPT because LL-RC incurs a higher cost. For LL-RC, only cache hits are plotted, as delayed hits do not apply to LL-RC. We do not observe any delayed hits for Online-DRL and OPT(Fig. 9c).

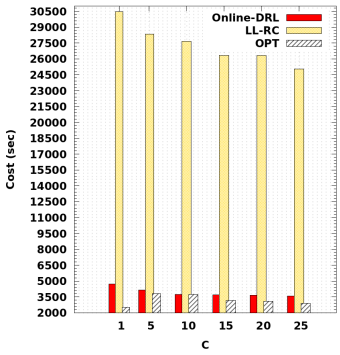
Experiment 10: This experiment examines the performance of Online-DRL, LL-RC, and OPT by varying the uplink bandwidth, focusing on cost, latency, hits, and delayed hits. Here, the cache size is fixed at $C = 5$. The results are shown in Fig. 10. In Fig. 10a, 10b,10c, for $x = 245760$, OPT has no optimal solution. The latencies obtained by all algorithms decrease with increasing uplink bandwidth, as then the time required to forward a service request reduces.

Experiment 11: Fig. 11 examines the performance of Online-DRL, LL-RC, and OPT as the downlink bandwidth is varied. Fig. 11a demonstrates that the cost of Online-DRL and LL-RC decreases with increasing downlink bandwidth because it takes less time to download services. In contrast, OPT exhibits a zig-zag pattern in cost; however, due to small variations, this pattern is not distinctly visible in the plot. In Fig. 11a, 11b,11c, for $x = 2375680$ and $x = 2457600$, OPT does not have an optimal solution. Fig. 11c indicates that hits increase with increasing downlink speeds, as with an increase in downlink bandwidth, a download condition ($T_i \geq M_i$) is more frequently satisfied. This in turn decreases latency.

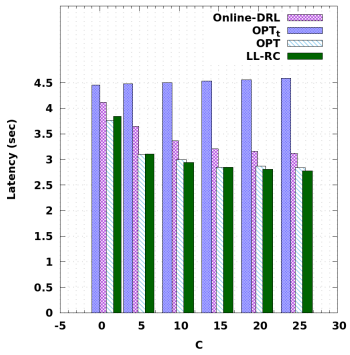
Experiment 12: This experiment compares the cost, latency, and hits for various request sizes. OPT has a pattern of varying cost in Fig. 12a. In Fig. 12a, 12b,12c, for $x = 0.2$, OPT has no optimal solution. The latency of Online-DRL (Fig. 12b) increases with increasing request sizes as more data have to be sent to the network. The latency for OPT and LL-RC is almost the same.

Experiment 13: Fig. 13 studies the impact of varying the maximum limits of C_{disk}, C_{cpu}, and C_{ram} are shown in Fig. 13a, Fig. 13b, and Fig. 13c. In this experiment, the total cache size C is set to 1100, which exceeds the number of service requests, allowing us to focus on the effects of varying the maximum limits for CPU, RAM and disk. As a result, the theoretical latency value is not calculated. The limit for each resource is set as the product of the x-axis value and the maximum value of the corresponding resource column. OPT performs the best in terms of cost, latency, and hits. In Fig. 13a, 13b,13c, for $x = 0.1$, OPT has no optimal solution. The latency of Online-DRL reduces when the resource limit is increased as more services can be cached.

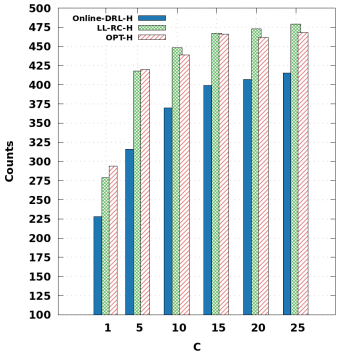
Experiment 14: Fig. 14 examines the impact of varying the value of θ on cost, latency and hits over a longer time horizon of 41347 rows from the SHARCNet dataset, with a cache size of $C = 10$. The cost of Online-DRL from Fig. 14a is very high for the value of $\theta = 0$ as the download conditions for both Online-DRL and LL-RC are the same. Fig. 14b shows that latency increases as θ increases. This is because the download condition ($T_i \geq \theta \cdot M_i$ or $L_i \geq \theta \cdot M_i$) is less frequently satisfied. However, the variation in latency is relatively small, making the increase less visible in the graph.



(a) Cost for various cache sizes

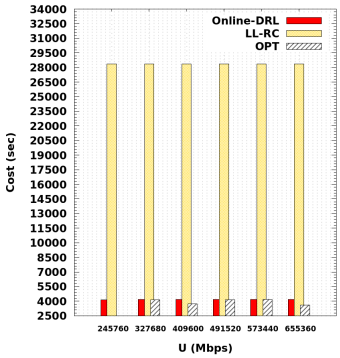


(b) Latency for various cache sizes

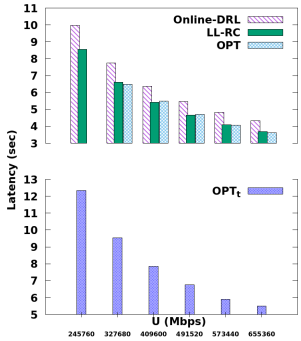


(c) Hits for various cache sizes

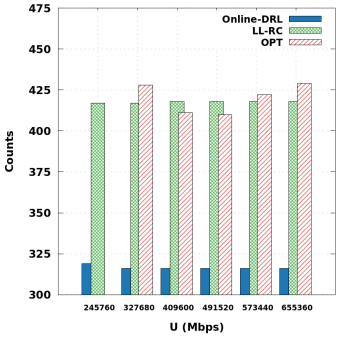
Fig. 9: Exp. 9: Online-DRL, LL-RC and OPT for a smaller time horizon, varying the cache size C.



(a) Cost for various uplink speeds

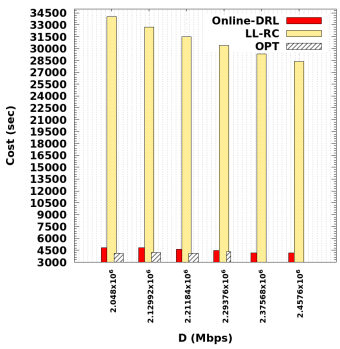


(b) Latency for various uplink speeds

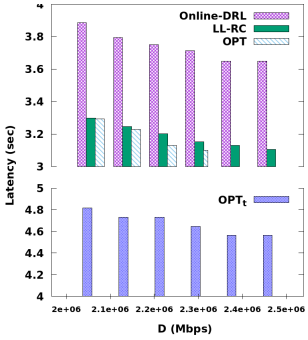


(c) Hits for various uplink speeds

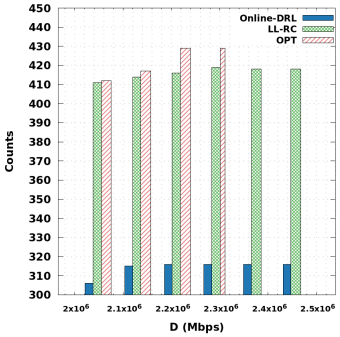
Fig. 10: Exp. 10: Online-DRL, LL-RC and OPT with various uplink speeds.



(a) Cost vs downlink speed



(b) Latency vs downlink speed



(c) Hits vs downlink speeds

Fig. 11: Exp. 11: Online-DRL, LL-RC and OPT for various downlink speeds.

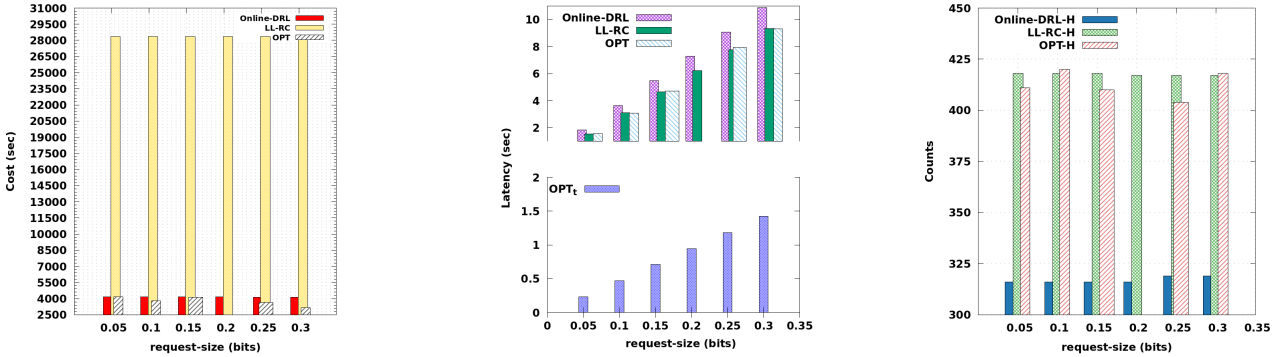
C. *Experiments using the synthetic dataset:*

We consider two parameters: ρ refers to the Zipf parameter indicating the skew of the distribution where higher values indicate higher skew, and ξ refers to the Poisson distribution parameter where higher values indicate more frequent requests for services. $\rho = 1.6$ and $\xi = 3 \times 10^6$ by default.

We vary C and plot latency, cost, hits, and delayed hits for different values of ρ (Fig. 15) and for different values of ξ (Fig. 16). For a given C, as the skew increases, the types of service requests that are repeated are reduced, but more of them satisfy the download condition, resulting in more downloads of services and fewer misses. These cause higher costs and lower latencies, respectively. As C crosses a certain value (near 5), repeated services do not need to be evicted, as many as before. This leads to a steeper fall in latency. Although delayed hits comparatively reduce for a higher value of ρ beyond $C = 5$, they do not affect latency, which continues to be low. When ξ increases, the request arrival rate increases. In that case, service requests arrive before the corresponding downloads are initiated or completed, resulting in higher misses (fewer hits), higher delayed hits, higher latencies, and lower costs.

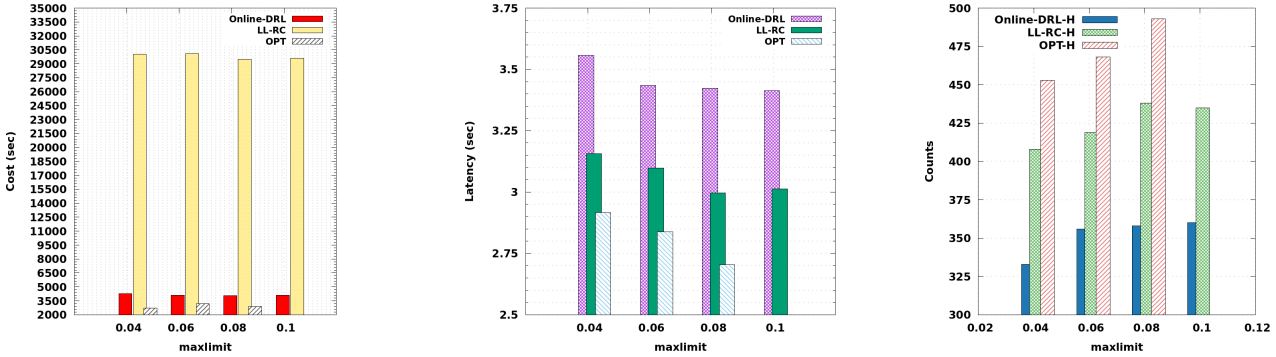
V. Discussion

Since there is a tradeoff between cost and latency, lower values of θ, where the latency is low and the cost is high, may be used in the context of consumer AR applications such as games and industrial AR applications such as AR-guided repair, which require very low latency. On the other hand, analytics applications such as traffic monitoring, package tracking, etc. can trade off latency for cost.



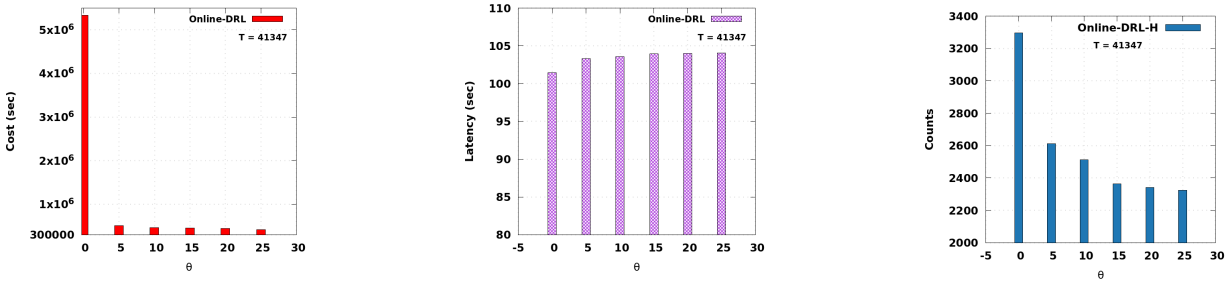
(a) Cost vs request size (b) Latency vs request size (c) Hits,Delayed hits vs request size

Fig. 12: Exp. 12: Online-DRL, LL-RC and OPT for various request sizes.



(a) Cost vs Max. Resource Limits (b) Latency vs Max. Resource Limits (c) Hits, Delayed hits vs Max. Resource Limits

Fig. 13: Exp. 13: Online-DRL, LL-RC and OPT for various maximum resource limits.



(a) Cost vs θ (b) Latency vs θ (c) Hits vs θ

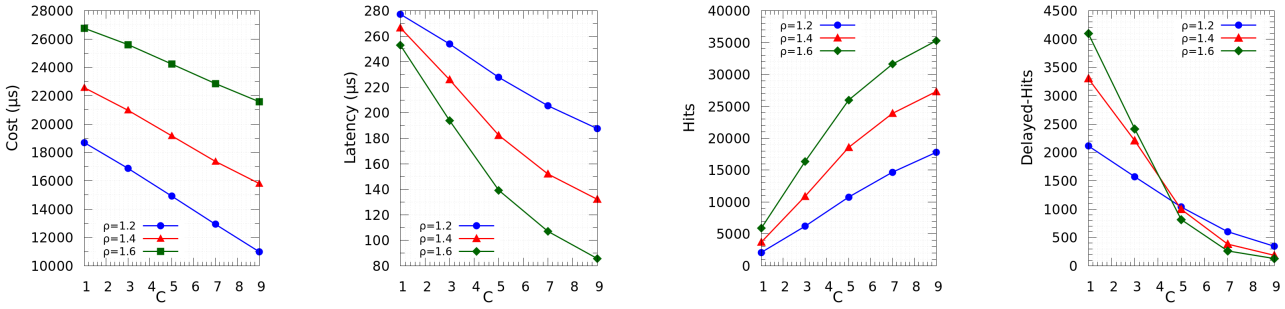
Fig. 14: Exp. 14: Online-DRL for varying θ for a bigger time horizon, T = 41347 entries.

Finding the time complexity of the current offline formulation (OPT) and possibly relaxing its constraints to improve the time complexity will be of theoretical interest and will be part of future work. In addition, we are working on extending this work to multiple edges and including chained services into the scope of the problem.

VI. Related Work

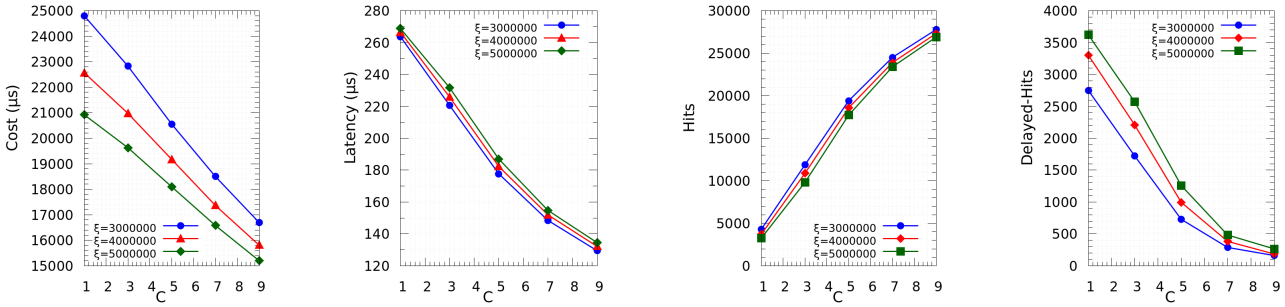
Service Caching: RED/LED [4] for service caching does not assume that service requests can arrive while a download is in progress or it assumes that downloads are instantaneous (proof for Theorem 3 [4]), both of which are strong assumptions. It also does not forward service requests to the cloud while a service is being downloaded, which may introduce high latency for large services. Their optimal algorithm downloads a service only if it is going to be requested M_i times in the future, thus optimizing only the cost, while our algorithm optimizes latency while keeping the cost below a maximum value. Fan et al. [6] discuss an online service caching algorithm with provably small caching regret under any service request arrival rate and do not consider delayed hits. The above papers do not consider caching semantics, while our paper does. SCRP [3] considers service replacement but does not consider download time and delayed hits.

Edge clouds may be rented to multiple service providers and these service providers may cache their services competing with each other. As more and more service providers start caching their services in edge clouds, the links from the edge become congested, causing deterioration of the service. An approximation algorithm that guarantees a Price-of-Anarchy is proposed in [17] and an online algorithm for the same problem in [18]. Service providers collaborate to share some services in another work by Xu et al. [19]. The delay experienced by an application is optimized considering both task offloading



(a) Cost vs. C (b) Latency vs. C (c) Hits vs. C (d) Delayed-Hits vs. C

Fig. 15: Exp. 15: Online-DRL varying the cache size C and varying ρ for synthetic datasets.



(a) Cost vs. C (b) Latency vs. C (c) Hits vs. C (d) Delayed-Hits vs. C

Fig. 16: Exp. 16: Online-DRL varying the cache size C and varying ξ for synthetic datasets.

and service caching by Xu et al. [20]. Tian et al. [21] use a distributed cooperative deep learning algorithm for service caching. Fan et al. [6] have the objective of finding an online service caching algorithm with provably small caching regret under any service request arrival rate. In this work, a request is served at the edge only if the service has already been downloaded and if there is sufficient processing power. This does not consider delayed hits. Moreover, we model the availability of CPU, RAM and disk to be associated with each service in cache.

Services may be partially cached at the edge, and resources at the edge may be rented for a period of time by clients. Prakash et al. [22] minimize the expected total cost of the system per time slot (the cost to forward the request to the cloud, to fetch the service and to rent it) when services are partially cached. A survey of service caching at the edge is also available [23].

Content caching: Atre et al. [24] are the first to consider delayed hits in content caching. Song et al. [25] propose relaxing Belady's algorithm and using machine learning techniques to improve latency in CDNs, but do not consider delayed hits or bypassing (that is, download the content, but do not cache it). Zhang et al. [14] propose an online general file caching algorithm that minimizes total request latency, that considers delayed hits and cache bypassing. Epstein et al. discuss [26] variants of online content caching that consider bypassing. If a file is not cached, the cache either fetches the file with a retrieval cost or rejects the file by not retrieving and caching it and paying a positive rejection penalty. The objective is to reduce the sum of both, thus generalizing both caching and caching with bypassing. Online caching on multiple caches with relaying and bypassing is also available [5], [27] where a request may be relayed to other caches. Gupta et al. discuss online caching in which the cache is variable in size and has a maintenance cost that depends on its size [28]. Li et al. [29] minimize the total latency considering delayed hits, relaying and bypassing. Most of the literature related to caching in MEC networks has been on content caching and these cannot be directly used for service caching. For more details, the reader is referred to two of the latest survey papers in this area [30], [31].

VII. Conclusions

Since edge caching reduces latency, the goal is to minimize total request latency within a time horizon while limiting cost and respecting resource constraints, without assuming request arrival patterns and considering delayed hits and request forwarding. In this paper, for the first time, we formulate this as an offline optimization problem (OPT) that minimizes latency under cost and resource constraints. We also propose an online algorithm, Online-DRL, for the same problem. We derive the competitive ratio of Online-DRL and evaluate it on Google and SHARCNet cluster traces, and a synthetic dataset. Our results show that Online-DRL is practical. Extending it to multiple edge caches is planned as future work.

VIII. Acknowledgements

This work was funded by the Science and Engineering Research Board (SERB), DST, Govt. of India, under the Project Code SPG/2021/002505.

REFERENCES

[1] R. Singh, R. Sukapuram, and S. Chakraborty, "Mobility-aware multi-access edge computing for multiplayer augmented and virtual reality gaming," in *NCA*, vol. 21. IEEE, 2022, pp. 191–200.

[2] Z. Xiao, Z. Xia, H. Zheng, B. Y. Zhao, and J. Jiang, "Towards performance clarity of edge video analytics," in *SEC*. IEEE, 2021, pp. 148–164.

[3] C.-K. Huang and S.-H. Shen, "Enabling service cache in edge clouds," *ACM Trans. on Internet of Things*, vol. 2, no. 3, pp. 1–24, 2021.

[4] 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 JSAC*, vol. 36, no. 8, pp. 1857–1870, 2018.

[5] H. Tan, S. H.-C. Jiang, Z. Han, and M. Li, "Asymptotically optimal online caching on multiple caches with relaying and bypassing," *IEEE/ACM Trans. on Netw.*, vol. 29, no. 4, pp. 1841–1852, 2021.

[6] S. Fan, I.-H. Hou, V. S. Mai, and L. Benmohamed, "Online service caching and routing at the edge with unknown arrivals," in *ICC*. IEEE, 2022, pp. 383–388.

[7] C. Zhang, H. Tan, G. Li, Z. Han, S. H.-C. Jiang, and X.-Y. Li, "Online file caching in latency-sensitive systems with delayed hits and bypassing," in *INFOCOM*. IEEE, 2022, pp. 1059–1068.

[8] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Commn. Surveys & Tutorials*, vol. 21, no. 3, pp. 2525–2553, 2019.

[9] H. A. Alameddine, M. H. K. Tushar, and C. Assi, "Scheduling of low latency services in softwarized networks," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1220–1235, 2019.

[10] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces: format + schema," 2014.

[11] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema, "The grid workloads archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, 2008.

[12] S. Deka and R. Sukapuram, "Edge service caching with delayed hits and request forwarding to reduce latency," in *2024 IFIP Networking Conference (IFIP Networking)*. IEEE, 2024, pp. 792–797.

[13] "Gurobi solver," 2025. [Online]. Available: http://gurobi.com

[14] A. D. Pia, S. S. Dey, and M. Molinaro, "Mixed-integer quadratic programming is in np," *Mathematical Programming*, vol. 162, pp. 225–240, 2017.

[15] N. E. Young, "On-line file caching," *Algorithmica*, vol. 33, no. 3, pp. 371–383, 2002.

[16] S. Fan, I.-H. Hou, and V. S. Mai, "Dynamic regret of randomized online service caching in edge computing," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[17] Z. Xu, Y. Qin, P. Zhou, J. C. Lui, W. Liang, Q. Xia, W. Xu, and G. Wu, "To cache or not to cache: Stable service caching in mobile edge-clouds of a service market," in *ICDCS*. IEEE, 2020, pp. 421–431.

[18] Z. Xu, H. Ren, W. Liang, Q. Xia, W. Zhou, G. Wu, and P. Zhou, "Near optimal and dynamic mechanisms towards a stable nfv market in multi-tier cloud networks," in *INFOCOM*. IEEE, 2021, pp. 1–10.

[19] Z. Xu, L. Zhou, S. C.-K. Chau, W. Liang, Q. Xia, and P. Zhou, "Collaborate or separate? distributed service caching in mobile edge clouds," in *INFOCOM*. IEEE, 2020, pp. 2066–2075.

[20] Z. Xu, S. Wang, S. Liu, H. Dai, Q. Xia, W. Liang, and G. Wu, "Learning for exception: Dynamic service caching in 5g-enabled mecs with bursty user demands," in *ICDCS*. IEEE, 2020, pp. 1079–1089.

[21] H. Tian, X. Xu, T. Lin, Y. Cheng, C. Qian, L. Ren, and M. Bilal, "Dima: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *World Wide Web*, pp. 1–24, 2021.

[22] R. S. Prakash, N. Karamchandani, V. Kavitha, and S. Moharir, "Partial service caching at the edge," in *WiOPT*. IEEE, 2020, pp. 1–8.

[23] C. Barrios and M. Kumar, "Service caching and computation reuse strategies at the edge: A survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–38, 2023.

[24] N. Atre, J. Sherry, W. Wang, and D. S. Berger, "Caching with delayed hits," in *SIGCOMM*, 2020, pp. 495–513.

[25] Z. Song, D. S. Berger, K. Li, A. Shaikh, W. Lloyd, S. Ghorbani, C. Kim, A. Akella, A. Krishnamurthy, E. Witchel *et al.*, "Learning relaxed belady for content distribution network caching," in *NSDI*, 2020, pp. 529–544.

[26] L. Epstein, C. Imreh, A. Levin, and J. Nagy-György, "Online file caching with rejection penalties," *Algorithmica*, vol. 71, pp. 279–306, 2015.

[27] H. Tan, S. H.-C. Jiang, Z. Han, L. Liu, K. Han, and Q. Zhao, "Camul: Online caching on multiple caches with relaying and bypassing," in *INFOCOM*. IEEE, 2019, pp. 244–252.

[28] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Panigrahi, "Elastic caching," in *SODA*. SIAM, 2019, pp. 143–156.

[29] G. Li, C. Zhang, H. Ni, and H. Tan, "Online file caching on multiple caches in latency-sensitive systems," in *CSoNet*. Springer, 2023, pp. 292–304.

[30] S. Safavat, N. N. Sapavath, and D. B. Rawat, "Recent advances in mobile edge computing and content caching," *Digital Communications and Networks*, vol. 6, no. 2, pp. 189–194, 2020.

[31] R. A. Dziyauddin, D. Niyato, N. C. Luong, A. A. A. Mohd Atan, M. A. Mohd Izhar, M. H. Azmi, and S. Mohd Daud, "Computation offloading and content caching and delivery in vehicular edge network: A survey," *Computer Networks*, vol. 197, p. 108228, 2021.

Sikha Deka is currently a Ph.D. scholar in the Department of Computer Science and Engineering at the Indian Institute of Information Technology Guwahati, India. She received her B.Tech and M.Tech degrees in Computer Science and Information Technology from Gauhati University, India. Her research interests include cloud computing, edge computing, service caching, and performance optimization in large-scale distributed systems. She actively contributes to academic research and has presented her work at international forums. She has also served as a teaching assistant for undergraduate networking and systems labs.

Radhika Sukapuram is a faculty member in the Department of Computer Science and Engineering at the Indian Institute of Information Technology Guwahati. Her research interests are in the area of Computer Networks and Distributed Systems. She received her Ph.D. in Computer Science from IIT Guwahati in 2018. She has industrial experience in mobile and telecom software and was a contributor to the GPRS specifications of the European Telecommunication Standards Institute. She received her B.Tech. in Electronics Engineering from National Institute of Technology Calicut in 1992, and her M.S in Software Systems from Birla Institute of Science and Technology, Pilani, in 1999.