Edge Service Caching with Delayed Hits and Request Forwarding to Reduce Latency

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

***Abstract*—Multi-access Edge Computing (MEC) networks reduce service latency by caching services at the network edge. A service request may be forwarded to the cloud for a response (Request Forwarding), contributing to latency. Additionally, the service may be downloaded, incurring a cost. If a download is ongoing, the request may be buffered if the remaining time to download is less than the time to forward the request and get a response (a Delayed Hit), also contributing to latency and not considered before. We argue that latency and cost must be considered separately, as the goal of MEC is reducing latency. For the first time, we pose the problem of minimizing the latency of service requests while limiting their costs, considering request forwarding, delayed hits, resource constraints and caching, and being agnostic to the request arrival pattern. We formulate this as an offline optimization problem. Since this is in NP, we propose an online algorithm. We experimentally demonstrate that this has better latency (upto 6.24%) and cost (upto 83.05%) across experiments using the Google cluster trace, relative to an online algorithm that downloads a service when it is requested if the service is not cached or if its download is ongoing.**

***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 in order to reduce the latency and network congestion incurred while accessing the cloud. A *service* processes inputs from devices and generates outputs, such as one for multiplayer AR games [1]. A service may be downloaded to the edge and cached to reduce latency.

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.

Recent work discusses caching of services considering the resource availability at the edge [2]. There are algorithms for service caching to minimize cost (defined as the sum of the downloading and forwarding times) [3], and those that consider relaying (sending the service request to nearby caches) and request forwarding [4]. Fan et al. [5] has the objective of finding an online service caching algorithm with provably small caching regret. *All the above discuss reducing or minimizing cost and do not consider delayed hits, to the best of our knowledge.*

A service request can be forwarded to the cloud and simultaneously a service download initiated. Since empirically the download cost is far higher than the cost of forwarding the service request [6], 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 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 the cloud and then to download the service. *Thus in service caching the total cost and the total latency are two different parameters* unlike in content caching [7]. *Since optimizing latency is an important requirement for low-latency edge services [8], we propose to minimize latency while keeping cost as a constraint.*

We address the following problem: 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 Delayed hits and Request forwarding, and with the objective of minimizing *Latency* and reducing cost? We call this the DRL problem. Our contributions are: 1) We formulate the offline version of DRL as an optimization problem and analyse its complexity. 2) We propose an online algorithm for DRL called Online-DRL 3) We state its competitive ratio to be $\Omega(Mk)$ where M is the time to download a service and k the maximum cache size, assuming a sufficiently large number of requests. 4) We evaluate Online-DRL using Google cluster traces and compare its latency and cost with the theoretical minimum latency 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. We demonstrate that the latency and cost of Online-DRL are upto 6.24% and upto 83.05% better than LL-RC.

II. System Model and Optimization Formulation

An edge cache consists of the set of services that are running. It stores the service id, the CPU, RAM and disk requirements of the service. C denotes the maximum number of services present in R. The variables used in the formulation are listed in Table I. The objective is to minimize the total latency of all service requests (eq.1). The total CPU (RAM,disk)

ISBN 978-3-903176-63-8 © 2024 IFIP

requirement of the items cached 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).

Since a service may be downloaded more than once, z_i' contains all the values of j_i at which download occurs (eq. 11). The optimization algorithm must choose a value of z_i such that it 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).

For the 1st request of any service s_i, there is no item s_i in cache (eq. 8). The decision to download is either 0 (download is not initiated) or 1 (eq. 3). 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.16). z_m is the earliest instance of s_i at which download is completed. After this (and before download is initiated), $y(s_i, z_i)$ continues to be 0 (eq.16). Let f_p be the pth service request (across services), represented by eq. 13. The service s_i is cached only after its download is completed (eq. 14). Hence the condition $(T(f_p) \geq (T(z_i) + M_i))$ is checked. The condition $(T(f_p-1) < (T(z_i)+M_i))$ is checked 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 latency is l_i. 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. 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 a maximum cost M_{max} (eq. 12).because the optimization program can choose to set $x(s_i, j_i)$ to 1 and evict whenever required.

$$min\ \Sigma_{i=1}^{U}\ \Sigma_{t=1}^{N}\ d(s_i, r(s_i, t)) \tag{1}$$
$$x(s_i, j_i) \in \{0, 1\} \quad \forall i \tag{2}$$
$$y(s_i, j_i) \in \{0, 1\} \quad \forall i \tag{3}$$
$$s.t.\ \ \Sigma_{i=1}^{N} S_i(1 - x(s_i, r(s_i, t))) \leq C_{disk} \quad \forall t \tag{4}$$
$$\Sigma_{i=1}^{U} CPU_i(1 - x(s_i, r(s_i, t))) \leq C_{cpu} \quad \forall t \tag{5}$$
$$\Sigma_{i=1}^{U} RAM_i(1 - x(s_i, r(s_i, t))) \leq C_{ram} \quad \forall t \tag{6}$$
$$\Sigma_{i=1}^{U}(1 - x(s_i, r(s_i, t))) \leq C \quad \forall t \tag{7}$$
$$x(s_i, 1) = 1 \quad \forall i \tag{8}$$
$$z_i \leq j_i \ \mid z_i \text{ has the maximum value possible } \forall i \tag{9}$$
$$z_i, j_i \in \mathbb{Z}\ \forall i \tag{10}$$
$$z_i \in z_i' \tag{11}$$
$$\Sigma_{i=1}^{U} y(s_i, z_i) * M_i < M_{max} \tag{12}$$
$$f_p \in \mathbb{Z} \tag{13}$$
$$x(s_i, A(s_i, f_p)) = 0,\ \forall i \text{ if } (T(f_p) \geq (T(z_i) + M_i)) \wedge (T(f_p - 1) < (T(z_i) + M_i)) \tag{14}$$

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

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

$$y(s_i, b_i) = \begin{cases} 1, \\ \quad \text{if} \quad b_i \in \{z_i, z_i + 1, \ldots, z_m\} \text{ where } T(z_i) \leq T(z_m) \leq (T(z_i) + M_i),\ \forall i \\ 0, \\ \text{otherwise} \end{cases} \tag{16}$$

The formulation is a Mixed Integer Quadratic Programming problem when implemented in Gurobi and is therefore in NP. Hence we propose an online algorithm called "Online-DRL".

III. Online algorithm for DRL

The main procedure of Algorithm 1 (line 5) checks if any services that are in the pending list to be downloaded have completed downloading. If yes, some items are evicted and the downloaded item is cached (line 10). If a request for a service arrives, it is processed (line 13). HANDLE_SERVICE_REQUEST() (line 13) must decide when to download a service, if it is not cached. After download starts, it

must decide whether subsequent requests for the same service must be buffered or forwarded to the cloud.

If a download for a service s_i is initiated the first time its request r_i arrives at time t_i, a request for s_i at $t_i + M_i$ units and later will not be missed. Therefore s_i is downloaded if T_i, 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 16 in Algorithm 1), is greater than or equal to M_i (line 19). If a download of s_i is not yet initiated and another request for s_i arrives before $t_i + M_i$ units of time, it will incur a latency of l_i if it is forwarded to the cloud. Thus if L_i, the sum of latencies of requests for s_i so far is greater than or equal to M_i, then too s_i is downloaded (line 19). T_i and L_i are calculated by the procedure GETPENALTY() in Algorithm 1. Both the retrospective downloads for s_i assume that there will be further 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 27), else it is forwarded to the cloud (line 29). If the request is a hit, its credit is set to M_i (line 31). If s_i is going to be cached, its cost is set to M_i (line 10).

Online algorithm based on Landlord [9]: In Algorithm 2, 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, items need to be evicted from cache (line 4). Each item has its credit initialised to its M_i when it is cached. After reducing the credit for each item (line 26), the set of files with credit as 0 are eligible for eviction. 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 of the sum and are 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), credits of 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.

The competitive ratio of Online-DRL is $\Omega(kM)$, where M is the maximum download cost and k is the maximum number of entries in cache. For want of space, the proof is skipped.

IV. Implementation and Evaluation

Datasets: The Google Cluster Traces [10] comprise Job and Task events files which are processed to create datasets the columns JobID, Timestamp (in microseconds), CPU, RAM, Disk, Task Index, and Status. Each unique JobID occurrence with distinct timestamps represents a service request and we consider only records with event type SUBMIT. Multiple datasets are generated, each including JobID, Timestamp, CPU, RAM, Disk, Task Index, and Status, with 0 values in CPU, RAM, or Disk fields replaced by the median value. The disk size is assumed to be in units of 1GiB.

Algorithm 1 Online algorithm to reduce latency: Online-DRL

1: **procedure** GETPENALTY(r_i,$tstamp$,l_i,$miss_i$)
2: $T_i = tstamp - miss_i$
3: $L_i = l_i$*Number of request forwards to the cloud from the time $miss_i$, including $miss_i$
4: return (T_i, L_i)
5: **procedure** MAIN
6: ▷ 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.
7: **while** True **do**
8: **for** each $s_i \in pending$ **do**
9: **if** s_i has completed download **then**
10: LANDLORD-RR(s_i,M_i,cpu,ram,$disk$)
11: Remove s_i from $pending$
12: $miss_i = -1$
13: **if** a request for a service arrives **then** HANDLE_SERVICE_REQUEST(s_i,l_i,r_i,$tstamp$,$miss_i$)
14: **procedure** HANDLE_SERVICE_REQUEST(s_i, l_i, r_i, $tstamp$, $miss_i$)
15: **if** s_i is not cached **then**
16: If $miss_i$ is -1, $miss_i = tstamp$ ▷ Time stamp of the very first miss or the first miss after an eviction
17: (T_i, L_i) =GETPENALTY(r_i,$tstamp$,l_i,$miss_i$)
18: **if** ($T_i \geq M_i$ or $L_i \geq M_i$) and $s_i \notin pending$ **then**
19: Initiate download
20: Add s_i to $pending$
21: Forward r_i to cloud ▷ A miss
22: **else if** ($T_i < M_i$ or $L_i < M_i$) and $s_i \notin pending$ **then**
23: Forward r_i to cloud ▷ A miss
24: **else**
25: **if** $s_i \in pending$ **then**
26: **if** remaining time to download $\leq l_i$ **then**
27: Buffer r_i ▷ A delayed hit
28: **else**
29: Forward r_i to cloud ▷ A miss
30: **else**
31: LANDLORD-RR(s_i,M_i,cpu,ram,$disk$) ▷ A hit

Evaluation: We compare Online-DRL with an algorithm that immediately downloads a service when it is 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.

Consider a service request s_i followed by M time slots

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 $items$

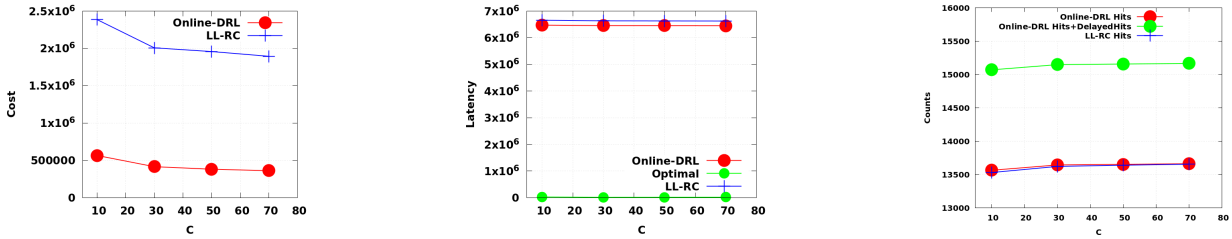
TABLE II: Parameters and their default values

Parameter	Description	Default value
l_{size}	Size of a forwarding request (or response)	1/10 * disk size median value
U	Uplink bandwidth	30 Mbits/s
D	Downlink bandwidth	40 Mbits/s
C	No. of items in cache	50
T	Time horizon (size of the dataset)	41349
C_{cpu}	Maximum edge CPU capacity	∞
C_{ram}	Maximum edge RAM capacity	∞
C_{disk}	Maximum edge disk capacity	∞

where there is no request, which is followed by M time slots with s_i requested in each of the M slots, followed by no requests in the next M slots. We call this a basic request. Consider k unique basic requests. Let the maximum number of entries in cache be k. When the first request of the basic request arrives, the optimal algorithm OPT forwards it to the cloud, incurring a latency of l and downloads and caches the service. The subsequent M requests of the basic request therefore incur 0 latency.Since OPT is an offline algorithm, apart from missing the first k basic requests, OPT can at most miss one basic request every k requests. Therefore the total latency of OPT for the n basic requests is at most $OPT_b = kl + \frac{n}{k}l$. The total latency due to Online-DRL is compared with the total theoretical optimal latency OPT_b. The rest of the metrics are compared with those of LL-RC. Since the algorithm is deterministic, we run each experiment only once. Experiments were run on a system with Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz and 8 GB RAM.

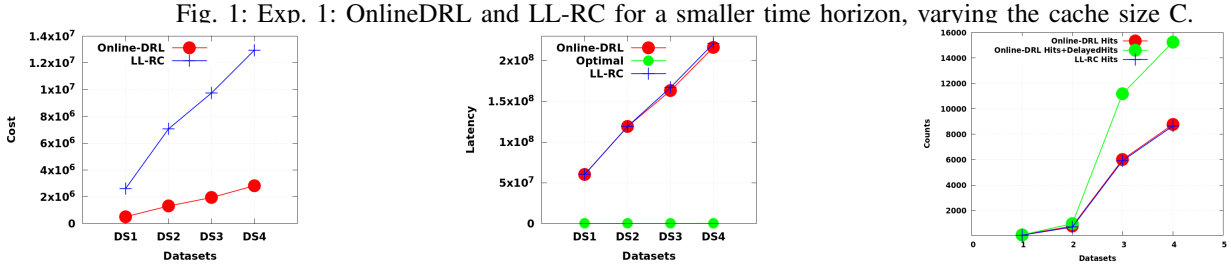
The results of *Experiment 1*, studying the impact of varying C for a given T are shown in Fig. 1a, Fig. 1b and Fig. 1c. In *Experiment 2*, the first 3 Google datasets are concatenated to create a new dataset A. We study the impact of varying T using *the same* data set A.From A the first 25000 ($DS1$), 50000 ($DS2$), 75000 ($DS3$) and 100000 ($DS4$) lines each are taken starting from the first entry each time. The results are in Fig. 2a, Fig. 2b, and Fig. 2c. The results of *Experiment 3*, studying the impact of varying U for a given T are in Fig. 3a, Fig. 3b, and Fig. 3c. The results of *Experiment 4*, studying the impact of varying D for a given T are in Fig. 4a, Fig. 4b, and Fig. 4c. The results of *Experiment 5*, studying the impact of varying l_size for a given T are in Fig. 5a, Fig. 5b, and Fig. 5c. The x-axis indicates the multiplication factor with the median value of the disk size of the dataset used. 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 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 b) the maximum resource requirement of any item in the dataset.

Conclusions: For all the experiments, we observe that Online-DRL performs better than LL-RC with respect to cost, latency and the sum of hits and delayed hits. The theoretical



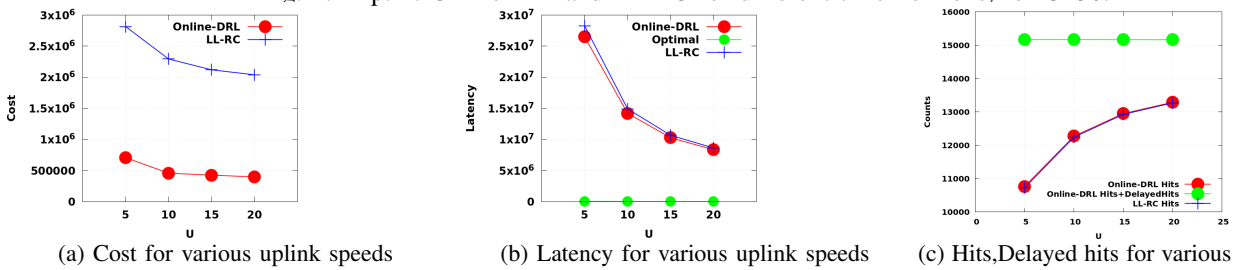
(a) Cost for various cache sizes (b) Latency for various cache sizes (c) Hits, Delayed hits for various cache sizes

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



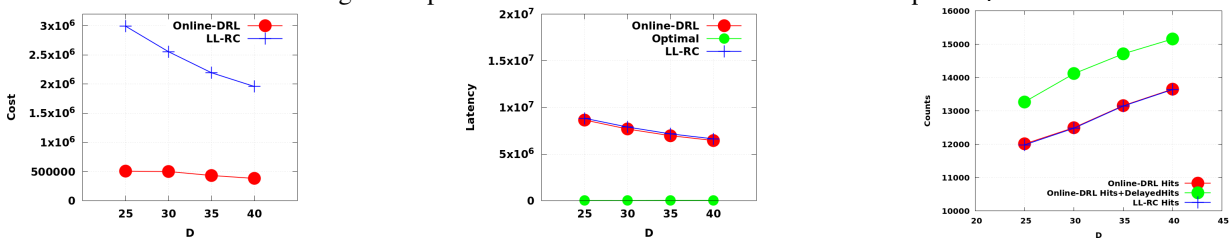
(a) Cost for increasing T (b) Latency for increasing T (c) Hits,Delayed hits for increasing T

Fig. 2: Exp. 2: OnlineDRL 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: OnlineDRL and LL-RC with various uplink speeds.



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

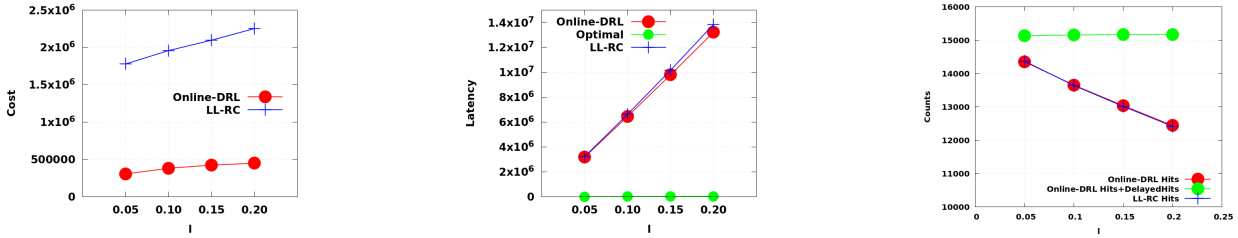
Fig. 4: Exp. 4: OnlineDRL and LL-RC for various downlink speeds.

minimum latency of the optimal offline algorithm is very low compared to the latency of Online-DRL, as expected.

V. Related Work

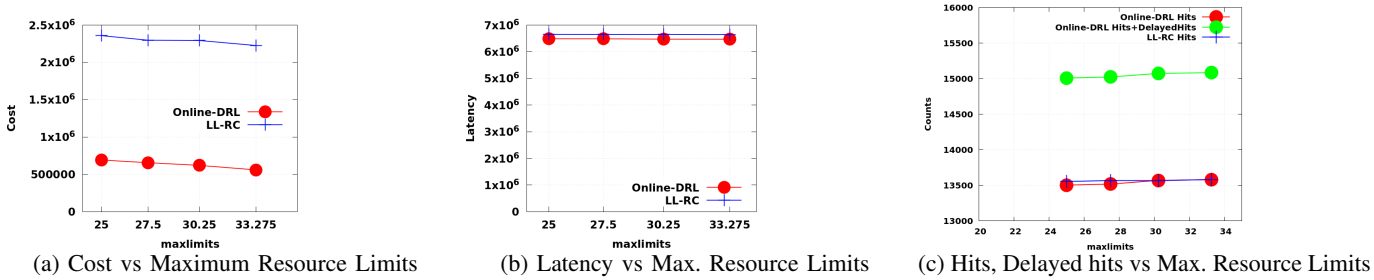
Service Caching: RED/LED [3] for service caching does not assume that service requests can arrive while a download is in progress or it assumes that downloads are instantaneous (proof for Theorem 3 [3]). It also does not forward service requests to the cloud while a service is being downloaded, which may introduce high latency for large services. Their optimal algorithm minimizes cost while ours optimizes latency while keeping the cost below a maximum value. Fan et al. [5] discuss an online service caching algorithm with provably small caching regret under any service request arrival rate and does not consider delayed hits. The above papers do not consider caching semantics while our paper does. SCRP [2] considers service replacement but does not consider download time and delayed hits.

Multiple service providers may cache their services competing with each other. An approximation algorithm that guarantees a Price-of-Anarchy is proposed in [11] and an online algorithm for the same problem in [12]. Service providers collaborate to share some services in another work by Xu et al. [13]. The delay experienced by an application is optimized considering both task offloading and service caching by Xu et al. [14]. A distributed co-operative deep learning algorithm is used by Tian et al. [15] for service caching. Fan et al. [5] has 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 serviced at the edge only if the service has already been downloaded and if there is sufficient processing power. This does not consider delayed



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

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



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

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

hits and we model the availability of CPU, RAM and disk to be associated with each service in cache.

Prakash et al. [16] 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 [17].

VI. Conclusions

When a service request arrives at an edge server, a decision needs to be taken on whether it needs to be only forwarded to the cloud (request forwarding) incurring latency or the corresponding service is to be downloaded from the cloud incurring cost, or both, or whether it needs to be buffered awaiting completion of download (a delayed hit), incurring latency. In this paper, for the first time, we formulated the above as an offline optimization problem with the objective of minimising latency with cost as a constraint. We proposed an online algorithm called Online-DRL for the above problem. This performs better in terms of latency (upto 6.24%) and cost (upto 83.05%) across experiments using the Google cluster trace, relative to an online algorithm that downloads a service when it is requested if the service is not cached or if its download is ongoing.

VII. Acknowledgements

This work was funded by the Science and Engineering Research Board (SERB), DST, Govt. of India, under the Project Code SPG/2021/002505. The authors are grateful to Amit Das for his review comments.

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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.