Reliable Cache Control in Unreliable Edge Environments Using Reinforcement Learning

Zheng Li
Graduate School of Information Science and Engineering
Ritsumeikan University
Osaka, Japan
gr0693ek@ed.ritsumei.ac.jp

Noriaki Kamiyama
Graduate School of Information Science and Engineering
Ritsumeikan University
Osaka, Japan
kamiaki@fc.ritsumei.ac.jp

***Abstract*—Mobile edge caching (MEC) deploys edge servers (ESs) at wireless base stations to cache popular contents close to users, reducing search latency, backhaul traffic, and requests to remote providers. However, because a large number of base stations must be equipped, many ESs are low-cost and less reliable, which increases unavailability due to failures. Existing work improves availability via erasure coding but typically assumes static pre-placement of contents, whereas practical ESs employ dynamic replacement (e.g., LRU). We propose a method that combines LRU-based dynamic replacement with erasure coding while accounting for ESs unavailability, and optimizes per-content cache insertion probabilities f_m via reinforcement learning (RL) so that the actual hit rate h_m approaches the target $\hat{h}_m$. Numerical experiments show that, compared with two baseline policies, including always caching upon a miss (fm = 1.0),our RL-optimized policy significantly improves the average successful recovery rate, i.e., the ratio of contents reconstructed using only chunks retrieved from ESs.**

***Index Terms*—Mobile edge caching (MEC), erasure coding, reinforcement learning, LRU, unreliable edge servers**

I. Introduction

Mobile edge caching (MEC) places servers close to end users to reduce latency and alleviate backhaul traffic, and has thus become an essential component of modern content delivery systems. In practice, MEC platforms deploy a large number of geographically distributed edge servers (ESs), many of which are low-cost and therefore subject to failures or temporary inoperability. As a result, ensuring content recoverability in the presence of unreliable ESs is a key challenge.

To address this issue, prior studies [6] have applied erasure coding to improve availability by distributing coded chunks across multiple ESs. However, these approaches rely on static pre-placement of contents and therefore do not reflect the dynamic replacement behavior of real MEC systems, which typically adopt online strategies such as LRU. How to incorporate redundancy control into dynamic cache replacement—while considering ES inoperability—remains an open problem.

This paper tackles this problem by integrating erasure coding with LRU-based dynamic replacement. We first translate the optimal redundancy levels obtained in [6] into per-content target hit rates under the LRU model. Then, we optimize the cache insertion probability f_m for each content using a lightweight reinforcement learning (RL) framework, which adjusts f_m so that the actual hit rate h_m approaches the target $\hat{h}_m$ despite ES failures.

Through extensive simulation evaluations, the following findings were confirmed:

- The proposed RL-based optimization successfully drives the actual hit rate h_m toward its target $\hat{h}_m$, enabling an appropriate configuration of f_m under LRU even when ESs are unreliable.
- When compared with two representative baselines—always caching upon a miss ($f_m(1.0)$) and a popularity-proportional probabilistic policy ($f_m(\mathrm{LHD})$)—the proposed method achieves a higher average content recovery success rate across a wide range of ES availability levels and popularity distributions.

II. Related Work

In MEC, a cooperative edge caching network is formed among neighboring edge servers (ESs) that are located in close proximity. By caching popular content on ESs near users, it becomes possible to reduce search latency and network traffic.

Guo et al. proposed a diversified cooperative caching scheme in which multiple edges collaborate to cache content. By retrieving content cooperatively among neighboring ESs in the event of a cache miss, they confirmed improvements in both overall hit rate and latency reduction [9]. However, these studies do not address detailed control considering ESs availability or the recovery threshold of erasure coding.

In recent years, distributed data storage systems capable of redundancy while maintaining high reliability have gained increasing attention [4]. The simplest redundancy technique is replication, where identical copies of the same content are stored across multiple nodes to enable recovery in the event of a failure. However, this approach suffers from low storage efficiency because each node must store a complete copy of the data.

To overcome this limitation, erasure coding has been widely adopted. In this method, data is divided into multiple fragments, and additional redundant fragments (erasure codes) are generated and distributed across multiple nodes. Even if some fragments are lost, the original data can be reconstructed. This technique enhances fault tolerance, reliability, and durability

under node failures or fragment loss, while achieving higher storage efficiency since complete duplication is not required.

Therefore, in order to address unreliable ESs, erasure coding can be utilized to enhance the reliability of cached files. However, maximizing the amount of data retrievable from unreliable ESs remains challenging for the following reasons.

Since the cache capacity of each ES is limited, there exists a trade-off between the reliability of cached files and the number of files that an ESs can store.

- Different files have different levels of popularity, and maintaining more popular files in the cache may provide greater benefits to users compared to less popular ones.
- The failure probability of ESs varies, and therefore the reliability of each file also depends on the locations of its chunks.

To address these challenges, it is necessary to appropriately determine the amount of redundancy added to each file, as well as the number and placement of chunks, under the constraint of limited cache capacity. Accordingly, a cache placement algorithm has been proposed for distributed cooperative edge caching systems that operate with unreliable resources [6].

In particular, Liu et al. [6] formulated a distributed cooperative caching system with unreliable edge resources and derived an optimal chunk allocation strategy under a static request model. Their work provides an offline redundancy design, but does not consider dynamic cache replacement such as LRU, which is widely used in practice. In this paper, we use only the optimal number of chunks per content obtained in [6] as a target redundancy level, and we do not repeat the algorithmic details of their dynamic programming method. The reader is referred to [6] for the full formulation and solution of the cache placement problem.

III. DEC System and Optimal Chunk Allocation

In unreliable edge caching environments, redundant chunk placement is essential to ensure that each content can be reconstructed even when a subset of edge servers (ESs) becomes unavailable. Liu et al. [6] formulated this problem as a distributed cooperative caching (DEC) system, where contents are encoded using an MDS code and the resulting chunks are distributed across ESs with limited capacity and non-zero failure probability.

A key modeling assumption in [6] is that each ES stores at most one chunk of a given content. This is consistent with the properties of MDS codes: placing multiple chunks of the same content on one ES does not increase recoverability, since the failure of that ES would simultaneously remove all colocated chunks. We adopt the same assumption to maintain consistency with the DEC model.

The cache placement problem (CP) in [6] determines the optimal number of chunks

$$y_m^*, \tag{1}$$

assigned to each content m, under homogeneous ES capacity and availability. The objective is to maximize the expected amount of recoverable data from ESs using a dynamic-programming-based algorithm.

In this paper, we do not reproduce the algorithmic details of the CP solver. Instead, we use only the optimal chunk counts obtained in [6] and define

$$T_m = y_m^*, \tag{2}$$

interpreting T_m as the *target expected number of chunks* of content m that should effectively reside on the ESs in our dynamic LRU environment. In [6], y_m^* is the optimal chunk allocation obtained by solving the HomoCP problem. We directly set $T_m = y_m^*$ and treat it as the target expected number of cached chunks in the dynamic LRU environment.

In the next section, we describe how the target chunk counts T_m are translated into target hit rates and subsequently realized by optimizing the content-wise insertion probabilities under LRU replacement.

IV. Proposed Method

We consider a cooperative edge caching system consisting of N homogeneous edge servers (ESs). Each ES has cache capacity s and availability rate r, and contents are cached under an LRU replacement policy. The demand for content m is modeled by a request rate q_m. Our goal is to determine the cache insertion probability f_m for each content m such that the actual hit rate h_m realized under LRU aligns with a target hit rate $\hat{h}_m$, which in turn corresponds to the optimal number of chunks T_m derived from the DEC formulation in Section III.

To improve readability, Table I summarizes the key symbols and parameters used throughout this paper.

A. Mapping Optimal Redundancy to Target Hit Rate

As discussed in Section III, we set $T_m = y_m^*$ as in (2) and interpret T_m as the target number of cached chunks of content m over all ESs.

In the homogeneous setting, each ES has availability rate r, and the number of ESs that are simultaneously available is modeled as a binomial random variable with parameters (N, r). If each available ES stores at most one chunk of content m, the expected number of *available* chunks of content m can be approximated as

$$\mathbb{E}[\text{available chunks of } m] \approx Nr\hat{h}_m, \tag{3}$$

where $\hat{h}_m$ is the steady-state hit rate of content m at an ES.

Note that f_m is an insertion probability rather than a residency probability. The steady-state presence of content m in an ES is given by h_m, not f_m. Therefore, the expected number of cached chunks is Nh_m (or Nrh_m when availability is considered), rather than Nf_m.

This approximation relies on the standard snapshot steady-state interpretation of the Che model: under the IRM assumption, the per-ES hit rate $\hat{h}_m$ also represents the steady-state probability that content m is resident in the cache at an arbitrary time instant. Since each ES can store at most one chunk of content m, the expected number of available chunks

TABLE I
NOTATION SUMMARY

Symbol	Description	Remarks
N	Number of edge servers (ESs)	All ESs are assumed homogeneous in capacity and availability.
s	Cache capacity of each ES	Measured in the number of chunks that can be stored at one ES.
r	Availability rate of an ES	Each ES is independently available with probability r.
M	Number of contents	Contents are indexed by $m \in \{1, \dots, M\}$.
k	Number of chunks required for recovery	A content can be reconstructed if at least k chunks are obtained.
y_m^*	Optimal number of chunks for content m in DEC	Obtained from the HomoCP solution in [6].
T_m	Target expected number of chunks of content m	Defined as $T_m = y_m^*$ and used as the redundancy target in our LRU-based model.
q_m	Request rate (popularity) of content m	Requests follow a Zipf distribution with parameter θ unless otherwise stated.
θ	Zipf parameter	Larger θ indicates a more skewed popularity distribution.
f_m	Cache insertion probability of content m	Decision variable optimized by the proposed method, subject to $0 \le f_m \le 1$.
h_m	Actual hit rate of content m at an ES	Steady-state hit probability under LRU with insertion probability f_m.
$\hat{h}_m$	Target hit rate of content m	Derived from T_m via the relation $T_m \approx Nr\hat{h}_m$.
t_C	Characteristic time of the LRU cache	Determined by the Che approximation as the solution to the fixed-point equation.
$L(f)$	Global loss function	Popularity-weighted squared error $L(f) = \sum_m q_m(h_m - \hat{h}_m)^2$.
e_m	Hit-rate error for content m	Defined as $e_m = h_m - \hat{h}_m$; its bins form the RL state space.
Δf	Step size for updating f_m	Actions are chosen from $\{-\Delta f, 0, +\Delta f\}$ in the Q-learning optimizer.
R_m	Reward associated with content m	Defined as $R_m = -q_m(e'_m)^2$, corresponding to the local contribution to $L(f)$.
α	Learning rate of Q-learning	Controls the update speed of the Q-values.
γ	Discount factor of Q-learning	Balances immediate and future rewards in the value update.
ε	Exploration rate in the ε-greedy policy	Gradually decayed from an initial value ε_0 to a minimum $\varepsilon_{\min}$.

is obtained by summing these residency probabilities over all ESs and multiplying by the availability rate r, which yields (3). By equating this expectation with the target chunk count T_m, we obtain

$$T_m = Nr\hat{h}_m, \tag{4}$$

which yields the target hit rate

$$\hat{h}_m = \frac{T_m}{Nr}. \tag{5}$$

Thus, the target hit rate $\hat{h}_m$ is a direct translation of the optimal redundancy level T_m computed in [6] into the LRU-based caching context.

B. Hit Rate Modeling under LRU

To relate the insertion probabilities $\{f_m\}$ to the hit rates $\{h_m\}$, we adopt the well-known Che approximation for LRU caching, which assumes an independent reference model (IRM) for content requests. Under this approximation, the hit rate of content m in a single LRU cache of capacity s is given by

$$h_m = 1 - e^{-f_m q_m t_C}, \tag{6}$$

where t_C is the so-called characteristic time, i.e., the time during which a content can remain in the cache without being requested before it is evicted.

The characteristic time t_C is determined by the cache capacity constraint via the fixed-point equation

$$\sum_{m=1}^{M} \left(1 - e^{-f_m q_m t_C}\right) = s, \tag{7}$$

which states that the expected number of distinct contents stored in the LRU cache equals its capacity s. Although Che et al. originally applied this model to hierarchical Web caching systems [7], the approximation in (6)–(7) has since been widely used as a general model for single-level LRU caches as well.

In our setting with N parallel ESs, each ES is modeled as having the same capacity s and following the same LRU policy. The hit rate h_m in (6) is interpreted as the per-ES hit rate for content m, which is then linked to the target number of chunks T_m through (4).

C. Optimization Objective

To align the realized hit rates with the target hit rates in (5), we define the following popularity-weighted loss function:

$$L(\mathbf{f}) = \sum_{m=1}^{M} q_m \left(h_m(\mathbf{f}) - \hat{h}_m\right)^2, \tag{8}$$

where $\mathbf{f} = (f_1, \dots, f_M)$, and $h_m(\mathbf{f})$ is given by (6) with t_C determined from (7). Minimizing $L(\mathbf{f})$ encourages contents with higher request rates to match their target hit rates more closely, which is desirable for improving the overall successful recovery performance.

This loss encourages contents with higher request rates to more accurately match their target hit rates, consistent with the goal of maximizing the overall success probability.

However, the optimization of (8) is challenging for the following reasons:

- The characteristic time t_C is defined implicitly by the nonlinear fixed-point equation (7), which couples all contents through the capacity constraint.
- The mapping from $\mathbf{f}$ to $h_m(\mathbf{f})$ is highly nonlinear.
- Analytical gradients of $L(\mathbf{f})$ with respect to $\mathbf{f}$ are difficult to obtain in closed form.

Therefore, a derivative-free search method is suitable for approximately solving this optimization problem.

D. Reinforcement Learning as a Derivative-Free Optimizer

Optimizing the insertion probabilities f_m by directly minimizing the loss $L(f)$ in (8) is difficult because the characteristic time t_C is implicitly defined by the nonlinear fixed-point equation (7), which couples all contents.

For each content m, the optimization variable f_m is adjusted through a small discrete action set $\{-\Delta f, 0, +\Delta f\}$. The resulting hit rate h_m is recomputed after updating f_m, and the deviation from its target, $e_m = h_m - \hat{h}_m$, is mapped to a small number of error bins that form the state space. The reward

$$R_m = -q_m(e'_m)^2 \tag{9}$$

represents the contribution of content m to the global loss $L(f)$, allowing Q-learning to act as a coordinate-wise optimizer. Although the Q-values follow the standard update rule, they are not interpreted as long-term MDP values but simply guide which local adjustment of f_m tends to reduce the weighted error.

Although the caching resource is shared through the common characteristic time t_C, the dependence among contents is already captured by the fixed-point equation (7). Therefore, updating one coordinate f_m at a time is equivalent to a coordinate-descent optimization of the global loss (8), while avoiding the complexity of optimizing an M-dimensional action space.

Because each update modifies only one coordinate while t_C is computed efficiently from a one-dimensional fixed-point equation, the resulting procedure is computationally lightweight and scales well with the number of contents. The method effectively searches for an f that aligns h_m with $\hat{h}_m$ without requiring explicit gradient computation.

V. Performance Evaluation

In this section, we compare the target hit rate $\hat{h}_m$ and the actual hit rate h_m for each content m, and verify that the proposed method can appropriately determine the cache insertion probability f_m. Furthermore, we evaluate two cases for the number of ESs (N) and compare the average successful recovery rate (the ratio of contents that can be reconstructed solely from chunks obtained from ESs) between the following three cases:

- f_m(RL): when f_m is optimized using the proposed reinforcement learning method, and
- $f_m(1.0)$: when content is always cached upon a cache miss.
- f_m(LHD): when f_m is assigned in proportion to the popularity q_m, following the Leave-Hot-Down caching rule.

We perform comparisons under varying ESs availability rates r, and also under different request rate distributions by changing the Zipf parameter θ.

A. Evaluation Conditions

The number of contents m is set to 50, and the minimum number of chunks required for content recovery k is set to 5. The availability rate r of each ES is varied from 0.7 to 0.9 in increments of 0.05. The cache capacity of each ES is $s = 6$, and the number of ESs N is set to 20 and 30. When $N = 20$, each can store 6 chunks, allowing a total of 120 chunks to be cached across all ESs. Contents are requested according to a Zipf distribution with parameter θ, and cache replacement is performed based on the LRU policy.

Reinforcement learning (RL) is performed using independent discrete Q-learning for each content. The number of episodes is set to 400, the learning rate to $\alpha = 0.5$, the discount factor to $\gamma = 0.9$, the initial ε-greedy exploration rate to $\varepsilon_0 = 0.3$, the decay rate to 0.995, and the minimum ε to 0.02. The step size for updating f_m is 0.02. The state space is defined by dividing the error $e_m = h_m - \hat{h}_m$ into five intervals: $(-\infty, -0.10]$, $(-0.10, -0.02]$, $(-0.02, 0.02]$, $(0.02, 0.10]$, and $(0.10, \infty)$.

B. Comparison Between Target Hit Rate $\hat{h}_m$ and Actual Hit Rate h_m

Fig. 1 shows the optimal cache insertion probability f_m obtained through RL for each content m, arranged in descending order of popularity (i.e., smaller m corresponds to higher popularity).

Fig. 2 compares the target hit rate $\hat{h}_m$ and the actual hit rate h_m in the same order. It can be observed that h_m approaches $\hat{h}_m$ under the proposed method, and the deviation between them is small. Highly popular contents have large request rates q_m, and thus can achieve a high hit rate even when f_m is not excessively large. Under the capacity constraint, however, f_m is suppressed as popularity decreases, and in the low-popularity region beyond a certain threshold, both $f_m \approx 0$ and $h_m \approx 0$ converge accordingly.

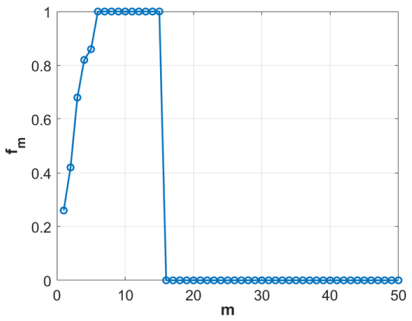

Fig. 1. Per-content f_m ($N = 20$, $r = 0.80$, $\theta = 1.0$).

C. Average Success Rate

Figures 3 and 4 show the average content acquisition success rate (q-weighted) for $N = 20$ and $N = 30$, respectively. Each plot compares the proposed method, where f_m is optimized by RL (f_m(RL)), with two baseline policies: (i) a strawman policy that always caches every content upon a cache miss ($f_m(1.0)$), and (ii) an LHD-type probabilistic caching strategy, where insertion probabilities are assigned according to content popularity. As the availability r of each ES increases, the failure probability decreases, and the average

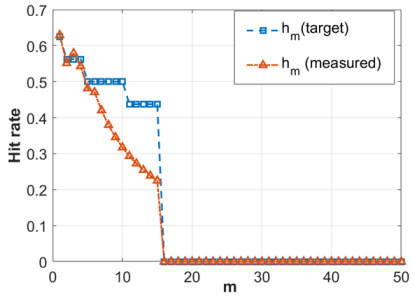

Fig. 2. Per-content hit rate ($N = 20$, $r = 0.80$, $\theta = 1.0$).

success rate improves for all methods. Compared with the naive baseline $f_m(1.0)$, the LHD policy already achieves a noticeable gain by preferentially allocating cache resources to popular contents. However, the proposed RL-based method further improves the average success rate over both baselines by learning a more fine-grained, per-content control of f_m.

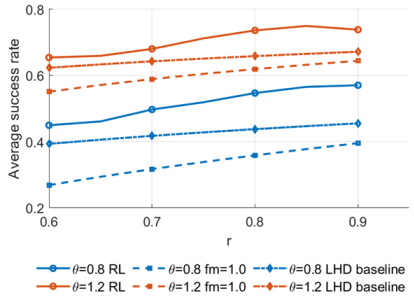

Fig. 3. q-weighted average success rate vs r (N=20): RL vs fm=1.0 vs LHD baseline

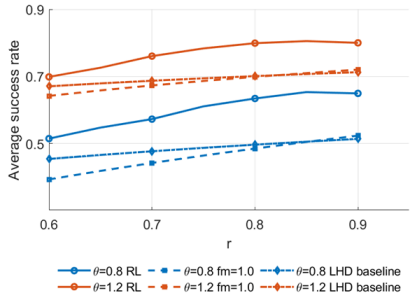

Fig. 4. q-weighted average success rate vs r (N=30): RL vs fm=1.0 vs LHD baseline

A slight non-monotonic decrease is observed in a narrow region (e.g., $r > 0.85$ in some settings). This phenomenon is not due to instability of the RL optimizer, but rather arises from the interaction between ES availability and request skewness. When θ is small, the popularity distribution becomes flatter, and the learned f_m concentrates more strongly on head contents. As r increases in this regime, the success probability of popular contents quickly saturates, while tail contents remain under-replicated and contribute little to the q-weighted objective. This imbalance may lead to a marginal decline in the overall average success rate. The 3D surface plot of success rate over (r, θ) (Fig.7) further confirms that this behavior appears only in a limited part of the parameter space, while the global trend remains increasing for commonly observed values of θ.

D. Effect of Fixed f

Using the insertion probability vector obtained in the training environment (hereafter referred to as fixed f^*), we evaluated the performance by varying the environmental conditions. Figures 5 and 6 show the transition of the average success rate with respect to the availability r and the Zipf parameter θ, as well as the consistency between the per-content distribution of f_m and the hit rate. In this subsection, comparisons with the baseline method are omitted, focusing only on the behavior of fixed f^*.

(a) Characteristics with respect to r: As the availability increases, the average success rate consistently rises, and a tendency of saturation is observed in the high-availability region. The transitions in the figures are smooth, showing stable behavior with little fluctuation. Even with fixed f^* alone, it can be observed that the performance naturally follows the improvement of resource availability.

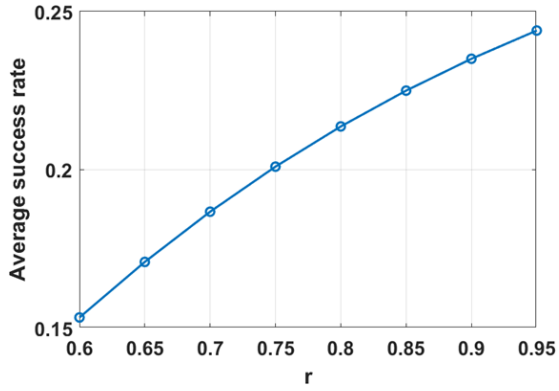

Fig. 5. Validation (f_m fixed): r sweep ($N = 20$, $\theta = 1.0$, training $r = 0.80$).

b) Characteristics with respect to θ: As the content popularity distribution becomes more concentrated toward the head (i.e., for larger θ), the overall average success rate increases. On the other hand, in the case of a more dispersed distribution (smaller θ), the rate of increase is limited, and the contribution of tail contents becomes relatively small. This reflects the fact that the allocation learned during training is consistent with the popularity ranking of the contents.

The per-content insertion probability f_m, when arranged in order of popularity, shows a pattern of being higher for highly popular contents and gradually decreasing as popularity declines. The corresponding hit rates are generally consistent with the intended target levels set during training, with little under- or over-allocation for the most popular contents. In the low-popularity region, a trade-off with the capacity constraint causes the hit rate to be suppressed, but overall, it shows a smooth decay trend.

Although the target redundancy levels T_m obtained from [6] are static, the proposed RL-based insertion policy generalizes

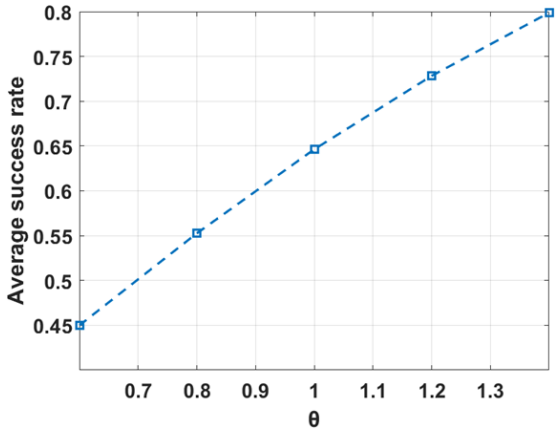

Fig. 6. Validation (fixed f_m): θ sweep (q-weighted) ($N = 20$, $r = 0.80$, training $\theta = 1.0$).

well under moderate popularity drift. As shown in Figs. 5 and 6, the learned insertion probabilities f_m^* exhibit smooth performance transitions when the Zipf parameter θ varies, indicating that the policy is not overfitted to a single popularity snapshot. Instead, the RL optimizer learns a robust allocation structure that remains effective across a range of request distributions. For environments with rapid or substantial popularity changes, the RL training procedure can be re-executed periodically at negligible computational cost.

E. 3D Success Rate Surface Over r and θ

Fig. 7 presents a 3D surface plot of the q-weighted average content acquisition success rate as a joint function of the ES availability r and the Zipf parameter θ. This visualization enables us to examine the interplay between server reliability and request skewness beyond the one-dimensional slices shown in Figures 3 and 4.

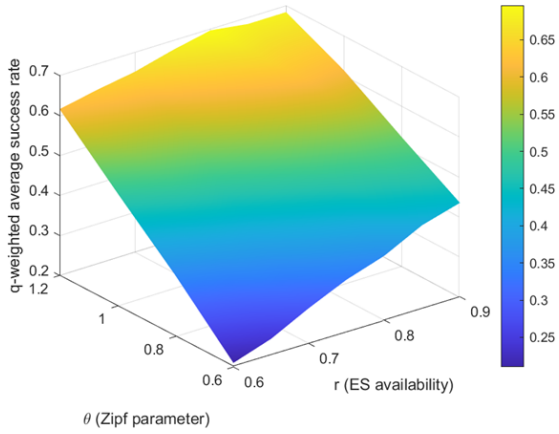

Fig. 7. SuccessRate(r, θ) ($N = 20$, $M = 50$, $s = 6$, $k = 5$)

As expected, the success rate generally increases as r becomes larger, because more available ESs lead to more opportunities for recovering the required k chunks. The surface also reveals that higher values of θ (i.e., more skewed popularity distributions) tend to improve overall performance, since the cache can focus more effectively on popular contents.

Importantly, the 3D surface clarifies that the slight non-monotonic dip observed in some 2D plots (e.g., for $r > 0.85$ under small θ) is confined to a narrow region of the parameter space. This effect is caused by saturation of popular contents combined with insufficient replication of tail contents, and is not due to instability of the RL optimization. Overall, the surface confirms that the global trend is smooth and increasing with respect to both r and θ.

VI. Conclusion

This paper presented a reinforcement-learning-based method for optimizing cache insertion probabilities in an erasure-coded MEC system with unreliable ESs. By mapping the optimal redundancy levels from [6] to target hit rates under LRU and adjusting f_m accordingly, the method successfully aligned actual hit rates with their targets.

Experiments showed that the proposed $f_m(\mathrm{RL})$ improves the content recovery success rate over two baselines—$f_m(1.0)$ and the popularity-proportional $f_m(\mathrm{LHD})$—under various ES availability and popularity distributions. Future work includes extending the method to handle real-time popularity drift and jointly optimizing redundancy and replacement.

ACKNOWLEDGEMENT

This work was supported by JSPS KAKENHI Grant Number 25K03113 and 23K28078.

References

[1] Y. Zeng, Y. Huang, J. Liu, and Y. Yang, "Privacy-preserving distributed edge caching for mobile data offloading in 5G networks," in *Proc. IEEE 40th Int. Conf. on Distributed Computing Systems (ICDCS)*, Singapore, 2020, pp. 541–551.

[2] L. Ramaswamy, A. Iyengar, and L. Liu, "Cache clouds: Cooperative caching of dynamic documents in edge networks," in *Proc. 25th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, Columbus, OH, USA, 2005, pp. 229–238.

[3] Y. Liu, X. Shang, and Y. Yang, "Joint SFC deployment and resource management in heterogeneous edge for latency minimization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 2131–2143, Aug. 2021.

[4] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[5] V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh, "Asymptotic interference alignment for optimal repair of MDS codes in distributed storage," *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 2974–2987, May 2013.

[6] Y. Liu, Y. Mao, X. Shang, Z. Liu, and Y. Yang, "Distributed cooperative caching in unreliable edge environments," in *Proc. IEEE INFOCOM*, London, U.K., 2022, pp. 1049–1058.

[7] H. Che, Z. Wang, and Y. Tung, "Hierarchical Web caching systems: Modeling, design and experimental results," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 7, pp. 1305–1314, Sep. 2002.

[8] L. Ramaswamy, L. Liu, and J. Zhang, "Efficient formation of edge cache groups for dynamic content delivery," in *Proc. IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, Lisboa, Portugal, 2006.

[9] Y. Guo, Y. Sang, B. Wang, and Y. Xu, "Collaborative diversified caching strategies for enhanced performance in edge server networks," in *Proc. IEEE ASENS*,Guangzhou, China,2025,pp. 282-285.