FACC: A Flexible and Asynchronous Updating Strategy for Cooperative Edge Caching

Zeming Gao*†
Beijing Univ. of Posts and Tele.
Beijing, China

Ye Tian*†
Beijing Univ. of Posts and Tele.
Beijing, China

Mengyu Yang†
Beijing Univ. of Posts and Tele.
Beijing, China

Edith C.H. Ngai
The University of Hong Kong
Hong Kong, China

Lanshan Zhang‡
Beijing Univ. of Posts and Tele.
Beijing, China

Wendong Wang†
Beijing Univ. of Posts and Tele.
Beijing, China

ABSTRACT

Recently, cooperative edge caching effectively reduces redundant transmissions and user plane latency by deploying potentially requested videos in edge nodes. Many existing works focused on designing accurate and diverse video selection strategies. However, they ignored that the edge nodes' heterogeneous requirements of replacement timings also impact the caching performance. In this paper, we propose FACC, a cooperative caching framework in which each edge node can flexibly select variable replacement timings and asynchronously update cached videos. We formulate the asynchronous cache replacement problem as an integer programming problem with a semi-Markov property. To solve this problem, we propose HiMAPPO, a multi-agent hierarchical deep reinforcement learning algorithm. It has been validated that FACC can effectively improve the edge hit rate and significantly reduce replacement costs compared with existing approaches on a real-world dataset.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**.

KEYWORDS

Mobile edge caching, Replacing strategy, Multimedia content delivery, Hierarchical MADRL

ACM Reference Format:
Zeming Gao, Ye Tian, Mengyu Yang, Edith C.H. Ngai, Lanshan Zhang, and Wendong Wang. 2024. FACC: A Flexible and Asynchronous Updating Strategy for Cooperative Edge Caching. In *Proceedings of The 3rd Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML 2024).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Cooperative edge caching broke down communication barriers among resource-limited edge nodes and pushed more abundant videos closer to users. It has become an emerging solution for multimedia **C**ontent **D**elivery **N**etwork (**CDN**), effectively reducing backhaul transmission and playback latency. To achieve the highest hit rate of the caching system, designing replacement strategies has become a research hotspot in recent years[12]. These studies mainly focus on accurate and diverse video selection with the considerations of user preference analysis[3], popularity prediction[6], and mobility prediction[10]. Many sensors are deployed at **M**obile **E**dge **C**omputing (**MEC**) servers to monitor and analyze real-time changes in user requests. Nevertheless, the appropriate replacement timing for each edge node is as significant to caching performance as the content selection. Unreasonable replacement time will prevent the edge nodes from making timely adjustments to meet the time-varying user requests, reducing the caching system's hit rate.

To adapt to user requests, some videos should be timely replaced at edge nodes when the user request patterns are suffering huge variations. We collected an extensive dataset of two-week video request traces from request sensors within a metropolitan area. As Fig. 1 shows, the variances of request patterns, which are the ratio of diverse requests in consecutive two hours, exhibit volatility in three random-selected edge nodes. These peaks can be considered as potential replacement timings, which demonstrate two characteristics: **randomness** and **asynchronism**. Furthermore, with the consideration of cooperation, selecting replacement timings for each edge node is a coupling and complex problem in cooperative edge caching. Existing studies are scarcely in-depth on the replacement timing, which is a critical decision.

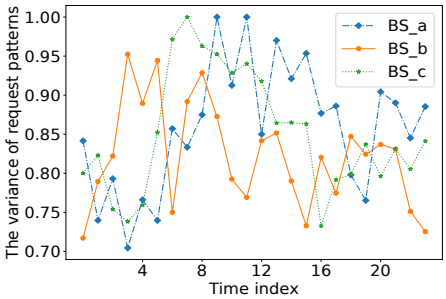


Figure 1: The variance in request patterns across three base stations, randomly selected, during each hour of the day.

These solutions can be roughly classified into two categories according to the replacement timing selections, namely, condition-triggered manner and fix-interval-based synchronized manners. Condition-triggered manner refers to caching a new video when certain pre-defined conditions are met. The conditions of passive caching strategies[4, 7] are defined as when a requested admissible video misses. Some studies[1] deploy a newly released video to edge nodes in advance, and the condition is that the video is predicted to

*Corresponding authors: zeminggao@bupt.edu.cn, yetian@bupt.edu.cn.
†The authors' institution is State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
‡The author's institution is Beijing Key Laboratory of Network System and Network Culture, Beijing University of Posts and Telecommunications

979-8-3503-6336-4/24/$31.00 ©2024 IEEE
DOI 10.1109/SenSys-ML62579.2024.00006

be popular. Otherwise, fix-interval-based synchronized replacement refers to all nodes simultaneously completing replacements in a unified frequency. Some studies [5] propose replacing videos for all nodes once a day during low network traffic, and others [8, 9, 14] specify the replacement interval of several hours.

These condition-based and fixed interval-based manners hurt the caching performance. On the one hand, these rule-based conditions cannot adapt to the randomness of user requests, and many inaccurate determines result in redundant caching and evictions, increasing unnecessary service costs. Furthermore, cache oscillations reduce the hit rate. On the other hand, a fixed interval may not be applicable to all nodes at all times. A too-short interval causes many myopic decisions, which reduces the hit rate.[13]. A too-long interval reduces the overall hit rate due to the lack of timely caching. Above all, a flexible and asynchronous collaborative caching mechanism is needed to meet the replacement requirements.

Selecting the accurate replacement timing is momentous in cooperative edge caching, and it brings three challenges. Firstly, it is necessary to extract the spatiotemporal information from the global request patterns to determine the optimal replacement time for each node. Secondly, the strategy for replacement timing selection needs to consider the requirements of adjacent edge nodes. Last but not least, the two questions must be rethought jointly: when is the best replacement timing, and what are the most appropriate replacement contents for each edge node? It complicates the solution space of the problem.

To this end, we propose a **F**lexible and **A**synchronous **C**ooperative **Ca**ching framework (**FACC**) that each edge node collaborates with its neighbouring nodes to decide individual replacement timings and contents. We introduce the Return On Investment (ROI) theory to measure the appropriate replacement timing. Meanwhile, we model the problem as a cooperative semi-Markov decision process (SMDP). To solve this problem, we propose a **Hi**erarchical Multi-Agent Proximal Policy Optimization (**MAPPO**) deep reinforcement learning algorithm (**HiMAPPO**). The high-level agents learn to select the optimal replacement timing for each edge node collaboratively. More specifically, the low-level agents, controlled by the high-level agents, collaborate to select the most appropriate replacement content for each edge node. After simulation with a real-world dataset, it shows that FACC can improve the hit rate by 4.7% and reduce the replacement cost by 53.4%, compared with the existing approaches.

2 FRAMEWORK AND SYSTEM MODEL

2.1 Flexible and Asynchronous Edge Caching Cooperative Framework

We assume that each area is served by only one combination of a base station (BS) and its external MEC server[14]. Each BS can flexibly replace cached contents at its replacement timing, and all BSs don't need to update simultaneously. Such an edge caching scenario for a citywide area is illustrated in Fig. 2. There are two replacing BSs which are getting new contents from a remote CDN server via the backbone network, and the other BSs reserve their cached contents. BSs are adjacent, and the served areas do not overlap. The external MEC server provides storage capacity and computing capability for content caching. Sequentially, a request is forwarded to the local BS, neighbouring BSs, and CDN server, and it will be served by anyone if the requested content is cached. All requested contents are already cached on the CDN server.

Table 1: The major notations of elements and sets.

Symbol	Implication
$j \in B$	The base station
$f \in F$	The content
$t \in T$	The time slot
N_j	The neighbour base station set of base station j
C_j	The caching capacity of BS j
s_f	The size of content f
j'	The neighbor base station
$z^t_{f,j}$	Whether BS j responded to the local request
$z^t_{f,j,j'}$	Whether BS j responded to the neighbor request
l, l'	The latency profit of local and neighbour hit
p, p'	The transmission profit of local and neighbour hit
$x^t_{j,f}$	The timing-decision of BS j at t
$y^t_{j,f}$	The content-selection-decision of f at BS j at t

The set of BSs is denoted as $B = \{0, 1, ..., j\}$, and C_j denotes the caching capacity of BS j. The set of contents is defined as $F = \{1, 2, ..., f\}$. Additionally, the size of the content f is denoted by s_f. Firstly, we posit that each BS can opt for replacement or retention at each predetermined discrete time slot $t \in T = \{0, 1, 2, ..., t\}$. We introduce an indicator function $x^t_j \in \{0, 1\}$, and x^t_j is set to 1 when BS j updates in time slot t, otherwise $x^t_j = 0$. The timing-decisions of all BSs is $X^t = \{x^t_0, x^t_1, ..., x^t_j\}$.

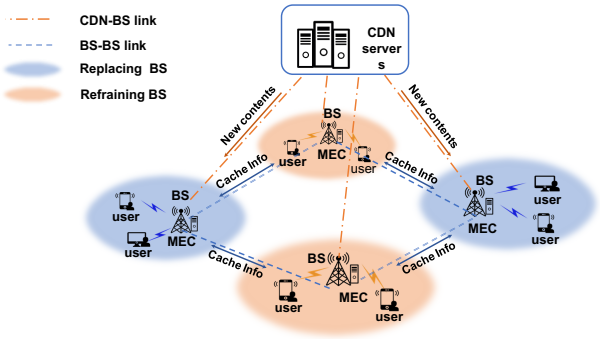


Figure 2: The FACC framework.

During the time slot t, the set of cached contents at BS j is $Y^t_j = \{y^t_{0,j}, y^t_{1,j}, ..., y^t_{f,j}\}$, which is the content-decisions of all BSs. Moreover, $d^t_{f,j}$ denotes the number of requests for f at BS j. These contents are used to serve requests from local BS and neighbouring BSs. To distinguish the served BSs, we introduce an indicator function $z^t_{f,j} \in \{0, 1\}$. $z^t_{f,j} = 1$ indicates that the request of f is served by local BS, otherwise $z^t_{f,j} = 0$. Each BS j has a certain number of neighbors $N_j = \{0, 1, ..., j'\}$. Similarly, another indicator function $z^t_{f,j,j'} \in \{0, 1\}$ indicates whether the BS j served the requests from the neighbouring BSs.

Compared to the CDN server, BSs hold the advantage of being closer to users, enabling faster delivery. The BSs get transmission profit and latency profit while serving user requests. The transmission profit of BS j is calculated as

$$T_j^t = \sum_{f\in F}\sum_{j'\in N_j} y_{f,j}^t z_{f,j}^t d_{f,j}^t s_f p + y_{f,j}^t z_{f,j,j'}^t d_{f,j'}^\tau s_f p' \tag{1}$$

where p and p' are the unit transmission profit when a BS serves a local request or a neighbour request. The video will start playing after the first segment is completely cached. The latency profit can be calculated as

$$L_j^t = \sum_{f\in F}\sum_{j'\in N_j} y_{f,j}^t z_{f,j}^t d_{f,j}^t l + y_{f,j}^t z_{f,j,j'}^t d_{f,j'}^t l' \tag{2}$$

where l and l' are the unit latency profit in case of local hits and neighbour hits. BSs get new content from the CDN server to improve service quality, which introduces additional transmission costs. So we express that c indicates unit replacement cost, the replacement cost of BS j during duration τ is calculated as

$$R_j^t = \sum_{f\in F} \max(y_{f,j}^t - y_{f,j}^{t-1}, 0) s_f c \tag{3}$$

2.2 Problem Formulation

In economics, ROI is the ratio of net profit to cost, which can measure both efficiency and effectiveness. This profit-cost consideration is matched with edge collaboration caching. To measure the efficiency of each update and the effectiveness of the newly cached contents, we introduce the ROI theory. By integrating Eq. 1 to Eq. 3 together, the **ROI** of **C**aching system (**ROC**) can be calculated as

$$\max_{X,Y} : ROC = \sum_{j\in B}\sum_{t\in T} \frac{\alpha T_j^t + \beta L_j^t - x_j^t R_j^t}{x_j^t R_j^t + \epsilon} \tag{4}$$

subject to :

$$\sum_{f\in F} y_{f,j}^t s_f \le C_j \tag{5}$$

$$z_{f,j}^t + \sum_{j'\in N_j} z_{f,j',j}^t \le 1 \tag{6}$$

where the indicator functions x_j^t and $y_{f,j}^t$ are optimization variables, and they ensure that the optimization problem is restricted to be solved in the binary range. Besides, α and β are weighted factors, and ϵ ensures that the denominator is not zero. Eq. 5 is the limit constraint on the cache capacity of each base station, Eq. 6 ensures that each request is responded by only one base station. And, t directly replaces τ in Eq.2-4. Such an integer programming problem has been proven to be NP-complete [9].

2.3 Semi-Markov Property

To clearly illustrate the nature of the problem, we introduce τ as another form of time slot t. As Fig.3 shows, BS j updates the cached videos after a flexible duration τ. The sequence of durations is denoted as $\Gamma_j = \{\tau_{j,0}, \tau_{j,1}, ..., \tau_{j,n}\}$ at BS j. Each flexible duration is composed of several consecutive time slots, which is described as $\tau_{j,n} = t - t'$, $t, t' \in T$ and subjected to $\sum_{\tau_{n,j}\in\Gamma_j} = |T|$. Hence, replacing the time slot t with duration τ, Eq.4 is transformed into

$$ROC = \sum_{j\in B}\sum_{\tau_{n,j}\in\Gamma_j} \int_{\tau_{n,j-1}}^{\tau_{n,j}} \frac{\alpha T_j^\tau + \beta L_j^\tau - R_j^\tau}{R_j^\tau}, n \ge 1. \tag{7}$$

Existing works[8, 9] assume that a BS makes caching decisions at fixed intervals. As shown in Fig.3, such a process is modelled as a MDP, believing that the caching state s transfers within each discrete time slot. However, in our framework, the duration τ of each decision is arbitrarily distributed rather than a fixed value. The caching state transition is related to the current state and its duration, so it only has Markov properties at the decision-making time but not at any time. Thus, the flexible replacement task can be characterized as a SMDP.

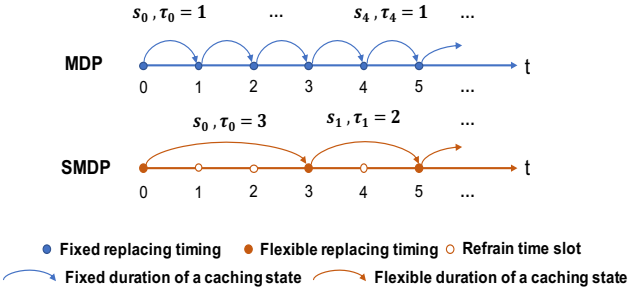


Figure 3: Caching decision-making processes of an edge node.

In a SMDP, the various durations of the states are controlled by additional decisions, which result in a high complexity of the model. To mitigate this challenge, one of the effective approaches is to employ hierarchical reinforcement learning (HRL). HRL divides a complex task into hierarchical subtasks, thereby reducing the complexity and improving learning efficiency. High-level subtasks control the state transition, and low-level subtasks make complex decisions. Given these advantages, we propose a hierarchical reinforcement learning based method to solve the problem.

3 COOPERATIVE EDGE CACHING SOLUTION BASED ON MULTI-AGENT HIERARCHICAL REINFORCEMENT LEARNING

In this section, we model the cooperative caching replacement problem with a partially observable semi-Markov process and present HiMAPPO, an option-based multi-agent hierarchical deep reinforcement learning algorithm.

3.1 Multi-agent Hierarchical Reinforcement Learning

For each BS, the flexible replacement task can be decomposed into two subtasks. The high-level subtask is to determine whether the current time slot is the best replacement time, and the low-level subtask is to select the most appropriate replacement contents. Most notably, the low-level subtask is controlled by the high-level subtask. For all BSs, their decisions should be cooperative. However, the observation of each BS contains local and neighbouring caching

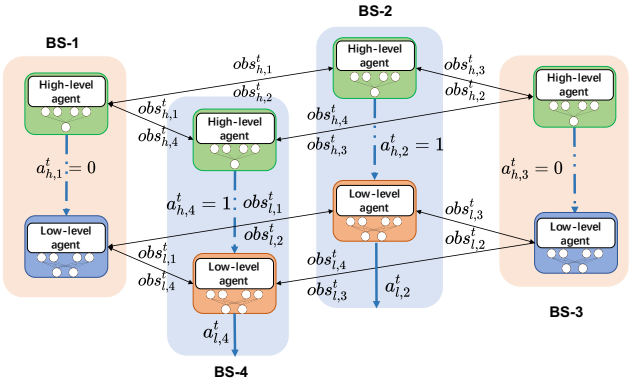


Figure 4: Multi-agent HRL framework.

information, which is a subset of the system. Therefore, the decision process of each BS is a Partially Observable Markov Decision Process (POMDP). These bring two challenges to the algorithm: hierarchical control and same-level collaboration.

To this end, we innovatively extend MAPPO into a hierarchical framework, HiMAPPO. To solve the problem presented in Eq. 4, we deploy two agents at a BS, and each is assigned to tackle a distinct subtask. As shown in Fig. 4, the high-level agent is akin to a switch which can control the low-level agent to select replacing contents. Agents at the same layer leverage their environmental observations to make informed caching decisions. In HiMAPPO, the heterogeneous agents undertake different subtasks and collaborate to make decisions and enhance the performance of the entire system.

3.2 High-level Agent Design

High-level agents have two options: initiate a replacement or refrain from doing so. They are used to measure the necessity of a replacement operation, with the objective of averting any potential degradation in caching performance. The high-level agents make decisions and get rewards within each discrete time slot, so such a decision-making process is a MDP.

High-level state design. Firstly, we define the partial observation of a high-level agent within time slot t as $obs^t_{h,j} = \{\boldsymbol{d}^t_j, \boldsymbol{y}^t_j\}$, where $\boldsymbol{d}^t_j = \{d^t_{1,j}, d^t_{2,j}, ..., d^t_{f,j}\}$ is a request pattern and the set of cached contents is $\boldsymbol{y}^t_j = \{y^{t-1}_{1,j}, y^{t-1}_{2,j}, ..., y^{t-1}_{f,j}\}$. Considering collaboration, the agents share observations with their neighbours. Therefore, the inputted state of a high-level agent is defined as $s^t_{h,j} = \{obs^t_{h,j}, \boldsymbol{obs}^t_{h,j'}\}$, where $\boldsymbol{obs}^t_{h,j'} = \{obs^t_{h,1}, ..., obs^t_{h,j'}\}$, $j' \in N_j$ are the shared neighbour observations.

High-level actions design. We define the action of a high-level agent as $a^t_{h,j} \in \{0, 1\}$, which represents caching new contents ($a^t_{h,j} = 1$) or not ($a^t_{h,j} = 0$).

High-level rewards design. Caching new contents improves the performance of the edge caching system but also incurs replacement costs. Another purpose of high-level agents is to reduce unnecessary replacements. To measure the necessity of a replacement, we define the reward for each agent as,

$$r^t_{h,j} = \frac{\alpha T^t_j + \beta L^t_j - x^t_j R^t_j}{x^t_j R^t_j + \epsilon} \tag{8}$$

where ϵ is a hyperparameter to ensure divisibility.

3.3 Low-level Agent Design

Low-level agents are used to select the most appropriate replacement contents under the control of corresponding high-level agents. They make decisions and get the rewards from their previous actions when the high-level decision is to replace. This decision-making process is a SMDP but has Markov properties at each decision-making moment.

Low-level state design. Different from high-level agent, low-level agent does not work in each time slot, and τ is the duration of the most recent replacement decision. So, the inputted state $s^t_{l,j}$ of a low-level agent contains all observations during duration τ, $s^t_{l,j} = \{obs^{t,\tau}_{h,j}, \boldsymbol{obs}^{t,\tau}_{h,j'}\}$, . And $obs^{t,\tau}_{h,j} = \{\boldsymbol{d}^{t,\tau}_j, \boldsymbol{y}^t_j\}$ is accumulation of recent observations, where $\boldsymbol{d}^{t,\tau}_j = \{\sum^t_{t-\tau} d^t_{1,j}, ..., \sum^t_{t-\tau} d^t_{f,j}\}$. The set of neighbours' observations $\boldsymbol{obs}^{t,\tau}_{h,j'}$ can be calculated as well.

Low-level action design. The action of a low-level agent is defined as $a^t_{l,j} = \{y^t_{1,j}, y^t_{2,j}, ..., y^t_{f,j}\}$, $f \in F$, where $y^t_{f,j} \in \{0, 1\}$. The number of selected contents is limited by C_j. If we model the action space as a multi-dimensional discrete action space, one-hot encoding cannot generate a certain number. To solve the problem, we use a multi-dimensional continuous action space. The output action space is defined as $\hat{a}^t_{l,j} = \{\hat{y}^t_{1,j}, \hat{y}^t_{2,j}, ..., \hat{y}^t_{f,j}\}$, $f \in F$, $\hat{y}^t_{f,j} \in [0, 1]$. The agent can select the most appropriate content by sorting.

Low-level reward design. The action of a low-level agent may be executed for several time slots, so its reward is calculated as,

$$r^t_{l,j} = \sum_{t-\tau}^{t} \alpha T^t_j + \beta L^t_j - K^t_j R^t_j \tag{9}$$

where τ is the duration of the most recent action. The practical significance of this reward is to improve the caching profit.

3.4 Network Architecture and Training

As shown in Fig. 5, all agents have the same network structures. The actor network (as θ) learns a caching policy, and the critic network (as ω) learns to predict total future rewards V_ω. Subsequent to the completion of an action, agents store the experiential data in a shared buffer. In order to capture the latent information of input, we add a long-short-term memory (LSTM) layer to each agent.

Subsequently, the learning model is trained in an experience-replay way. When the replay buffer is full, all agents synchronously proceed to update the network parameters utilizing experiential data of completed decisions. Obviously, low-level agents have different amounts of experiential data. To solve this challenge, we stipulate that a low-level agent stores the most recent replacement experiential data in a replay buffer when the high-level action $a^t_{h,j} = 0$. These marked data will not be used in training. Besides, the experiential data will be stored normally when $a^t_{h,j} = 1$. To prevent excessive convergence of high-level agents, we randomly initialized the cached content at fixed time intervals.

To maximize total rewards, we train each agent with a policy gradient method. Firstly, we calculate the advantage function by

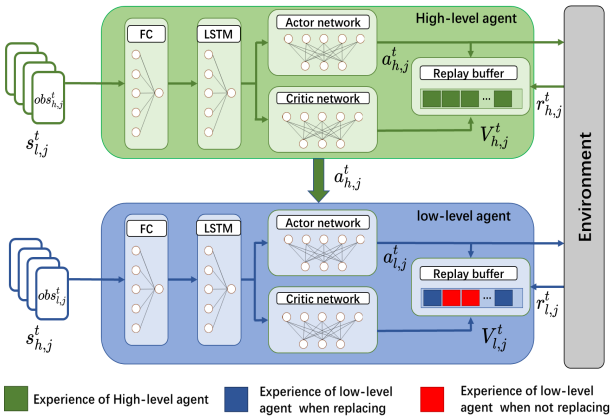


Figure 5: The learning network architecture of an edge node.

the generalized advantage estimator method as,

$$A(a_{h,j}^t|S_h^t) = \sum_{l=0}^{T} (\gamma\lambda)^l (r_{h,j}^t + \gamma V_\omega(S_h^{t+1}, \omega_{h,j}) - V_\omega(S_h^t, \omega_{h,j})) \tag{10}$$

where γ, λ is discount factor. $S_h^t = \{s_{h,1}^t, s_{h,2}^t, ..., s_{h,j}^t\}$ is high-level state of system, and $j \in B$. Then, we update the actor network as,

$$\theta_{h,j} = \arg\max_\theta (\frac{1}{T}\sum_{t=1}^{T} \min[imp_{h,j}^t A(a_{h,j}^t|S_h^t), clip(imp_{h,j}^t, 1+\varepsilon, 1-\varepsilon)A(a_{h,j}^t|S_h^t)] - \sigma H(\pi_\theta)) \tag{11}$$

where T is the length of replay buffer, ε is the hyperparameter of the clip method[11], $H(\pi_\theta)$ is the entropy of policy π_θ, σ is a hyperparameter to control the entropy value, and $imp_{h,j}^t$ is importance sampling function, which is calculated as,

$$imp_{h,j}^t = \frac{\pi_{\theta_h}(a_{h,j}^t|obs_{h,j}^t)}{\pi_{\theta_h'}(a_{h,j}^t|obs_{h,j}^t)} \tag{12}$$

To update the high-level critic network, we use the gradient descent method to solve the following equation:

$$\omega_{h,j} = \arg\min_{\omega_h} (\frac{1}{B\cdot T}\sum_{t=0}^{T}\sum_{j=0}^{B} \max[(V_{\omega_h}(S_h^t) - r_{h,j}^t)^2, (clip(V_{\omega_h}(S_h^t), V_{\omega_h'}(S_h^t) - \varepsilon, V_{\omega_h'}(S_h^t) + \varepsilon) - (r_{h,j}^t)^2]) \tag{13}$$

The low-level agents are updated in the same method as the high-level agents, so we do not describe them.

4 PERFORMANCE EVALUATION

4.1 Dataset construction

The dataset mentioned in section 1 comprises approximately 4.2 million requests, recording the video ID, timestamp, user ID, and user IP of each request. We designated a subset of adjacent IP addresses serviced by a specific network operator. This subset was partitioned into 100 logical areas, each composed of IP addresses exhibiting the highest degrees of adjacency[2].

For evaluation, 17 edge areas are randomly selected from the real-world dataset. Only the most frequently requested videos are considered, and the content number is 600. We did not obtain detailed information about the video, Without loss of generality, we variously set the size of each video content to $[10, 50]$ MB.

4.2 Setup and Metrics

Each BS has the same number of neighbours, which is set to 4. The transmission distance is set to 1 hop between neighbour BSs, and 5 hops between CDN and BSs. Because the latency is positively related to the distance, we assume that the latency is 100 ms when a request is served by the CDN server and 10 ms by neighbour BSs[8]. To balance the transmission hops and latency, α and β are set to 1 and 2. The hyperparameter ϵ is set to the replacement cost of a video. Particularly, the initialized cache set is stuffed with the least popular contents.

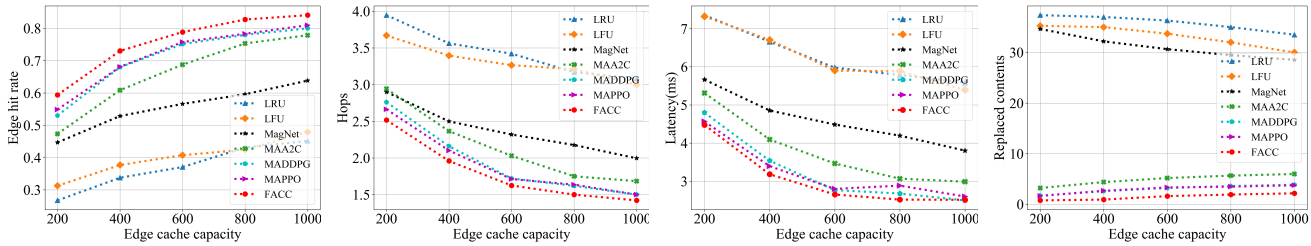
We implement the HiMAPPO learning model using pytorch, which runs on a server with RTX 3090 GPU, Intel Xeon Gold 5222 3.8 GHz CPU and 32 GB memory. The actor and critic network learning rates of the high-level agents are set to 2e-4 and 1e-4, and the learning rates of low-level agents are set to 5e-4 and 2e-4. We set γ and λ as 0.99 and 0.95 by default and the size of the hidden layers is 1024. The dataset for training includes a request trace in 12 days, and the leftover trace is used for evaluation. In parallel, the models were trained for 10e5 episodes with 16 random seeds, and the best-performing model was applied to performance evaluation.

The following metrics are considered to evaluate the caching performance of each mechanism. 1) Average edge hit rate: This fraction of requests with local and neighbour hits over the total requests. 2) Average transmission hops: The average hops between the responded server and the user of all requests. 3) Average transmission latency: The average latency of all requests in the user plane. 4) Average replaced contents: The average replaced contents for each BS per hour.

4.3 Evaluation result

FACC is compared with two types of baseline mechanisms which are mentioned above. The condition-triggered mechanisms include LRU, LFU, and MagNet[7]. The fix-interval-based synchronized mechanisms include MAA2C [9], MADDPG[8], and MAPPO, in which MAPPO is a simplification of FACC with only the low-level agents. The fixed interval is set to 4 hours.

First, We consider the impact of cache capacity and set the capacity of each BS to 200, 400, 600, 800 and 1000 MB. As shown in Fig. 6, we can find that FACC has a better caching performance than the condition-triggered mechanisms. Compared with MagNet, which is one of the advanced condition-triggered mechanisms, FACC drastically improves the edge hit rate and reduces both the transmission hops and latency by 20%, 24%, and 29%. It is due to that FACC accurately recognize the replacement timings and prevents the cache oscillations. In comparison to the fixed-interval manners, FACC provides a flexible replacement interval for each BS so that timely adjustments can be made to meet the time-varying user requests. Thus, FACC has an average improvement of edge hit rate by 7%, reduces latency by 10%, and hops of transmission by 13%.



(a) Average edge hit rate (b) Average transmission hops (c) Average latency (d) Average replaced contents

Figure 6: The caching performance of different mechanisms.

From Fig. 6d, we can find that FACC significantly reduces the number of replaced contents. Compared with the condition-triggered mechanisms, FACC has an obvious reduction of redundant caching and eviction, which reduces the replaced contents by more than 90%. In the same way, FACC has an average replacement reduction of 67% than the fix-interval-based mechanisms.

Compared with MADDPG, one of the state-of-the-art cooperative caching strategies, the edge hit rate of FACC is higher than MADDPG by 4.7%. Indeed, the replaced contents of FACC is 61% significantly lower. Furthermore, we change the discrete decision-cycle t of FACC when the capacity is set to 600 MB. As Fig.7 shows, with the cycle increasing, the hit rate of FACC declines. When the decision cycle is also set to 4 hours, FACC and MADDPG have similar hit rates, but FACC maintains the least amount of replaced contents because of the accuracy of replacement timing. It shows that FACC can effectively determine the appropriate replacement timings and achieve good caching performance with minimal replacement cost. Above all, FACC implements a joint solution for both timely replacement timing and accurate replacement content.

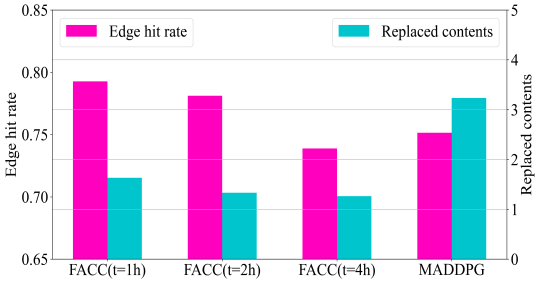


Figure 7: Caching performance of FACC at different cycles.

5 CONCLUSION

In this paper, we proved the heterogeneous requirements of replacement timings for edge nodes through large-scale tracking analysis. Thus, we proposed FACC, a cooperative edge caching framework with flexible and asynchronous replacement timings. Correspondingly, we implemented an intelligent algorithm HiMAPPO to solve the collaborative SMDP. It has been validated that FACC can ensure high quality of service and reduce replacement costs. Our work compensates for the lack of flexibility in cooperative caching frameworks.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62072047, and in part by the National Natural Science Foundation of China under Grant 62072048.

REFERENCES

[1] Bahman Abolhassani, John Tadrous, Atilla Eryilmaz, and Edmund Yeh. 2022. Fresh caching of dynamic content over the wireless edge. *IEEE/ACM Transactions on Networking* 30, 5 (2022), 2315–2327.

[2] Yong Cui, Jian Song, Minming Li, Qingmei Ren, Yangjun Zhang, and Xuejun Cai. 2017. SDN-based big data caching in ISP networks. *IEEE Transactions on Big Data* 4, 3 (2017), 356–367.

[3] Baotian Fan, Yanxiang Jiang, Fu-Chun Zheng, Mehdi Bennis, and Xiaohu You. 2022. Social-aware Cooperative Caching in Fog Radio Access Networks. In *ICC 2022 - IEEE International Conference on Communications*. 1672–1677.

[4] Yu Guan, Xinggong Zhang, and Zongming Guo. 2021. PrefCache: Edge Cache Admission With User Preference Learning for Video Content Distribution. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 4 (2021), 1618–1631.

[5] Chunlin Li, Mingyang Song, Chongchong Yu, and Youlong Luo. 2021. Mobility and marginal gain based content caching and placement for cooperative edge-cloud computing. *Information Sciences* 548 (2021), 153–176.

[6] Sajad Mehrizi, Saikat Chatterjee, Symeon Chatzinotas, and Björn Ottersten. 2020. Online Spatiotemporal Popularity Learning via Variational Bayes for Cooperative Caching. *IEEE Transactions on Communications* 68, 11 (2020), 7068–7082.

[7] Junkun Peng, Qing Li, Xiaoteng Ma, Yong Jiang, Yutao Dong, Chuang Hu, and Meng Chen. 2022. MagNet: Cooperative Edge Caching by Automatic Content Congregating. In *Proceedings of the ACM Web Conference 2022*. 3280–3288.

[8] Manoj Kumar Somesula, Rashmi Ranjan Rout, and Durvasula VLN Somayajulu. 2022. Cooperative cache update using multi-agent recurrent deep reinforcement learning for mobile edge networks. *Computer Networks* 209 (2022), 108876.

[9] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun. 2020. Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2499–2508.

[10] Lin Yao, Xiaoying Xu, Jing Deng, Guowei Wu, and Zhaoyang Li. 2022. A Cooperative Caching Scheme for VCCN With Mobility Prediction and Consistent Hashing. *IEEE Transactions on Intelligent Transportation Systems* 23, 11 (2022), 20230–20242.

[11] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems* 35 (2022), 24611–24624.

[12] Yuchao Zhang, Pengmiao Li, Zhili Zhang, Bo Bai, Gong Zhang, Wendong Wang, and Bo Lian. 2019. Challenges and chances for the emerging short video network. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1025–1026.

[13] Yuchao Zhang, Pengmiao Li, Zhili Zhang, Bo Bai, Gong Zhang, Wendong Wang, Bo Lian, and Ke Xu. 2020. AutoSight: Distributed Edge Caching in Short Video Network. *IEEE Network* 34, 3 (2020), 194–199.

[14] Yiwei Zhao, Ruibin Li, Chenyang Wang, Xiaofei Wang, and Victor CM Leung. 2021. Neighboring-aware caching in heterogeneous edge networks by actor-attention-critic learning. In *ICC 2021-IEEE International Conference on Communications*. IEEE, 1–6.