Transient Data Caching Based on Maximum Entropy Actor-Critic in Internet-of-Things Networks

Yu Zhang
School of Computer and Electronic Information
Guangxi University
Nanning, China
zhangy.03@qq.com

Ningjiang Chen*
School of Computer and Electronic Information
Guangxi University
Nanning, China
chnj@gxu.edu.cn

Siyu Yu
School of Computer and Electronic Information
Guangxi University
Nanning, China
gaiusyu6@gmail.com

***Abstract*—With the rapid development of the Internet of Things (IoT), a massive amount of transient data is transmitted in edge networks. Transient data is highly time-sensitive, such as monitoring data generated by industrial devices. Due to their inefficiency, traditional caching strategies in edge networks are inadequate for handling transient data. Thus, to improve the efficiency of transient data caching, we construct a freshness model of transient data and propose a maximum entropy Actor-Critic based caching strategy, TD-MEAC—which can improve the freshness of cached data and reduce the long-term caching cost. Simulation results show that the proposed TD-MEAC achieves a higher cache hit rate and maintains a higher average freshness of cached transient data compared with the existing DRL and baseline caching strategies.**

Keywords—Internet of Things; edge networks; transient data caching; maximum entropy Actor-Critic

I. Introduction

The Internet of Things (IoT) development has caused exponential device growth and generated vast data at the edge network. The data generated by IoT devices are used in extensive applications across various domains. Most IoT data is transient, with short-lived, time-sensitive, and dynamic characteristics. For example, some health monitoring sensors produce specific health parameters only valid for a few minutes. The invalid data can affect the reliability and accuracy of application decisions.

Caching transient data at the edge network can accelerate response times and reduce network traffic, thereby improving Quality of Service (QoS) and Quality of Experience (QoE)[1]. For instance, in a vehicle traffic monitoring system, collecting and timely updating the status data of the monitoring objects is crucial. Caching this valid status data at the edge nodes eliminates the need for IoT devices to respond to data requests, thereby reducing the load on the return path. Additionally, for some latency-sensitive applications, such as intelligent power distribution monitoring systems, responding to requests by edge nodes can more quickly isolate device failures[2].

Most edge caching studies only apply to data that remains valid in the long term. Researchers [3]-[6] propose corresponding caching schemes for different caching scenarios suitable for long-term valid data. However, due to IoT data's transient nature, the data gradually becomes obsolete and invalid.

The main challenges transient data poses to existing caching strategies are summarized as follows:

- Data freshness. The caching of transient data needs to consider data freshness, which depends on the application's timeliness and lifecycle requirements. Data becomes stale with time, resulting in freshness loss. Thus, caching transient data requires balancing caching benefits and freshness loss.
- Frequent replacements. Compared to long-lived data, the caching of transient data requires more frequent cache replacements, which may increase the energy consumption of edge nodes.

Therefore, caching strategies for long-lived data are not suitable for transient data.

Some studies have sought to improve traditional caching strategies to adapt to the transient data. A cache replacement strategy based on data freshness is proposed in [7], which selects data with the shortest remaining lifespan for replacement when cache replacement occurs. Reference [8] suggested the expected validity of data lifecycles, which predicts the number of requests received by cached data in the remaining lifespan and evicts data with the lowest expected number of requests for cache replacement. References [9][10] investigate transient data caching in vehicular networking and transient data caching on routers, respectively. And [11] used the least recently used replacement strategy and considered the transient nature of the data to determine whether to cache it by setting a freshness threshold.

Although these methods consider data transience, they require prior knowledge, such as popularity and network topology. However, edge networks are dynamic, with the position, connection status, resource distribution, application scenarios, and demands of IoT devices constantly changing, making it challenging to provide prior knowledge.

Deep reinforcement learning (DRL)[12] has recently attracted widespread attention as a new machine learning paradigm. DRL uses deep neural networks to represent and process high-dimensional state space, improving decision-making capabilities in large-scale state space. Without prior knowledge and state features of edge networks, DRL can generate a series of mappings between network states and caching actions through multi-round interactions with edge networks, making DRL an excellent method for solving the caching problem of transient data in edge networks.

In [13], DRL was first used to solve the caching problem of transient data in edge networks. An Actor-Critic method is trained in an asynchronous and parallel manner, allowing caching decisions to be made without a priori knowledge.

DOI reference number: 10.18293/SEKE2023-159.

Researchers [14] proposed a hierarchical network architecture caching system, formulating the problem as a Markov decision process (MDP) and designing a policy optimization solver to obtain the optimal policy. Reference [15] considered the limited caching capacity and used a distributed proximal policy optimization (DPPO) algorithm to optimize the allocation of cached data and improve training speed.

Unlike the above methods, this paper proposes a transient data cache strategy based on the Maximum Entropy Actor-Critic, TD-MEAC, which is based on the traditional Actor-Critic algorithm and considers the randomness of cache actions. It improves the exploration ability and can better adapt to the dynamic edge environment. The main contributions of this paper are as follows:

- A caching strategy for transient data is proposed, considering the data lifecycle and propagation delay to construct a freshness model for the data. A cost function that balances the freshness loss of the data and the cost of acquiring data items is also presented, which can reduce the long-term acquisition cost of transient data.
- Considering the practical limitations of the caching scenario, the caching replacement problem for transient data is formulated as MDP. The freshness of data items is viewed as a part of the state space, and the efficiency function is applied to the reward function. The Actor-Critic algorithm with maximum entropy interacts with the environment and finds the optimal caching decision.

This paper compares the TD-MEAC with other approaches, including classical and DRL-based caching strategies. Simulation results show that the proposed TD-MEAC achieves a higher cache hit rate and maintains a higher average freshness of cached transient data.

II. System Model

A. Network Architecture

This paper considers a centralized caching scenario for a group of IoT devices covered by edge nodes in edge networks, as shown in Fig. 1. This scenario consists of IoT devices, applications, and edge nodes.

Edge nodes randomly distribute N IoT devices within their coverage area, with each IoT device capable of generating only one data item type. The corresponding edge node can cache these data items. When an IoT application sends a data request, the edge node checks its cache unit to see if there is a related data item. If there is, and the freshness of the data item meets the requirements, the edge node responds to the request. Otherwise, the edge node forwards the request to the corresponding IoT device, which generates a new data item and answers via the return link.

B. Freshness model of data items

Freshness reflects the staleness of transient data, and a lower freshness indicates a more stale data item. Worn data items can reduce the accuracy and timeliness of applications. Different applications have additional requirements for data freshness. Factors that affect data freshness include data collection and processing time, transmission delay, and timeliness of data processing.

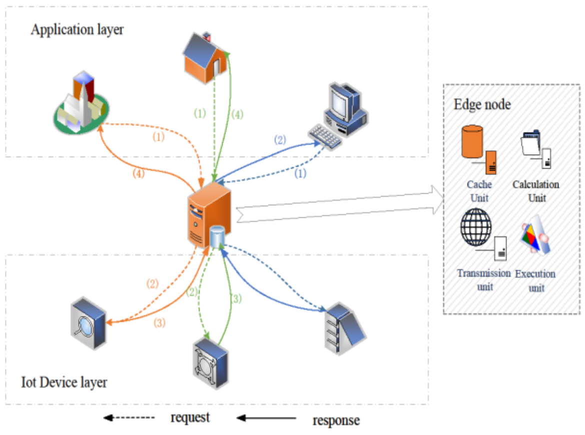

Fig. 1 The network architecture of edge caching

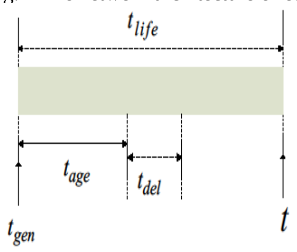

Fig. 2 data item is valid

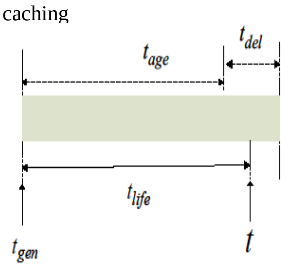

Fig. 3 data item is invalid

A static content identifier *CID* uniquely identifies the IoT data items. IoT data item i has a timestamp field t_{gen}^{i} and a lifecycle field t_{gen}^{i}. For a given time t, the age of the data item i is defined as $t_{age}^{i} = t - t_{gen}^{i}$. The response delay of the edge node t_{del}^{i} consists of two parts: the delay of caching and retrieval after receiving the data request and the propagation delay of the data item from the edge node to the IoT application. The freshness of the data item i, denoted as $freshness_i$, is defined as follows:

$$freshness_i = \begin{cases} \dfrac{t_{life}^{i} - t_{age}^{i} - t_{del}^{i}}{t_{life}^{i}} & ,t_{life}^{i} > t_{age}^{i} + t_{del}^{i} \\ 0 & ,otherwise \end{cases} \quad (1)$$

As shown in Fig. 2, the data item is still valid and can be used to respond to the request. In contrast, in Fig. 3, the cached data item has expired, and the carried information is invalid, thus cannot be used as a response to the request.

C. Cost Function

The caching of transient data at edge nodes results in a lag, causing the cached data to have a non-zero data age compared to directly acquired data from IoT devices. Consequently, the cache results in a loss of data freshness. The cost of freshness loss for a newly generated data item from IoT devices can be considered zero, while the cost of freshness loss for a cached data item i is related to its data age. Therefore, the cost of freshness loss denoted as $Cost_{loss}$, is defined as follows:

$$Cost_{loss} = \begin{cases} \dfrac{t_{age}^{i}}{t_{life}^{i}} & ,t_{age}^{i} < t_{life}^{i} \\ 0 & ,otherwise \end{cases} \quad (2)$$

The acquisition cost of data denoted as $Cost_g$, depends on where the data is obtained. The cost of acquiring data from an edge node is denoted as $Cost_{g1}$, while the cost of acquiring data from IoT devices is denoted as $Cost_{g2}$. The generation of data items by IoT devices consumes device energy, and there is a transmission delay from IoT devices to edge nodes. Thus, it is generally true that $Cost_{g1}<Cost_{g2}$.

When an edge node responds to a request, the cost consists of two parts: the cost of freshness loss $Cost_{loss}$, and the cost of acquiring data items by the edge node $Cost_{g1}$. When IoT devices respond to the request, there is only the cost of acquiring data items $Cost_{g2}$. Both costs affect the benefit of caching transient data. To balance the two costs, the cost function, denoted as *Cost*, is defined as follows:

$$Cost = \psi Cost_{loss} + \varrho Cost_g, \quad \psi + \varrho = 1 \tag{3}$$

Where ψ and ϱ respectively denote the weighting coefficients for the cost of freshness loss and the acquisition cost of data items.

To evaluate the effectiveness of caching, we define the benefit function *U*. Minimizing the cost is equivalent to maximizing the benefit since the cost of caching is inversely proportional to its benefit. To ensure that the benefit function is always non-negative, we define the constant $\alpha Cost_{g2} + \beta$ as the baseline benefit value. Therefore, the benefit function is defined as follows:

$$U = \psi(Cost_{g2} - Cost_{loss}) + \varrho\left(1 - Cost_g\right) \tag{4}$$

III. Problem formulation

A. Cache Replacement Model

The cache state of edge nodes is determined solely by the previous cache state and the previous caching action. Therefore, the cache replacement process can be represented as MDP. Typical MDP consists of a quintuple $\{S, A, P, R, \gamma\}$. In the transient data cache model, the number of requests processed by an edge node in a single time step is unknown. If the unknown number of requests is regarded as part of the MDP, the complexity of the model will increase exponentially. Therefore, this paper regards processing a single request by an edge node as a decision cycle and replaces the time step with a decision cycle. And the quintuple for the *n*-th decision cycle can be represented as $\{s_n, a_n, p(s_{n+1}|s_n, a_n), r_n, \gamma\}$.

The state space *S* represents a finite set of states for the caching process of edge nodes. The state features are split into two parts: the data items already cached in the cache space and the requests from IoT applications. The cache space size is defined as *C*, and *C* binary tuples can represent the data item features $\hat{d}_i = \{CID_i, freshness_i\}$ .The request is represented by $\hat{d}_0 = \{CID, requestTime\}$. Thus, the state S_n for the *n*-th decision cycle can be represented as $s_n = \{\hat{d}_0, \hat{d}_1, \hat{d}_2 \dots \hat{d}_c\}$.

The action space *A* represents a finite set of caching actions for edge nodes. The action taken by the edge node in the nth decision cycle is represented by a_n. The size of the action space is proportional to the cache space and is related to the selected caching action. The larger the action space, the greater the computational complexity of the model. In resource-constrained edge nodes, to limit the size of the action space, it is stipulated that at most one cache replacement occurs in a decision cycle. The action space is represented as $A = \{0,1,2,3, \dots, C\}$. Here, $a_n = 0$ represents skipping the cache replacement process while $a_n = 1 \dots c$ represents replacing the new data item with the a_n-th data item in the cache space.

The state transition probability $P(s_{n+1}|s_n, a_n)$ represents the probability of the following decision cycle state S_{n+1} given the state-action combination $\{s_n, a_n\}$.

The reward function *R* is an instantaneous utility function. r_n represents the expected instantaneous reward for the state-action combination in the *n*-th decision cycle. The utility function *U* serves as a suitable indicator for training DRL, balancing freshness loss cost and data acquisition cost. Define reward function *R* as:

$$R(s_n, a_n) = E\left[r_n | s_n, a_n \sim P\right], where\ r_n = U \tag{5}$$

In addition to the immediate reward, the impact of future rewards on the current decision is also considered. A discount rate γ is used to reconcile the cumulative rewards of the reward sequence. The smaller the γ is, the more biased the immediate reward is. Define the cumulative rewards G_n starting from the *n*-th decision cycle as:

$$G_n = \sum_{i=0}^{+\infty} \gamma^i r_{n+i} \tag{6}$$

The goal of MDP is to find an optimal caching policy π^* that maximizes the long-term reward under this policy, as shown below:

$$\pi^* = argmaxE\left[G_n\right] \tag{7}$$

B. TD-MEAC: Transient Data Caching Strategy Based on Maximum Entropy Actor-Critic

To enhance the exploration capability of the strategy, the randomness of cached actions is considered when selecting them, and the maximum entropy policy is used to improve the solution to the problem. The improved optimal policy π^* is defined as:

$$\pi^* = argmaxE\left[G_n + \alpha \sum_{i=0}^{+\infty} \gamma^i H\left(\pi\left(\bullet | s_i\right)\right)\right] \tag{8}$$

Here, $\pi(\cdot\,|s_i)$ represents the probability distribution of cached actions taken under the marginal node state s_i. The function $H(\pi(\cdot\,|s_i))$ represents the entropy of the cached actions, which measures the randomness of the caching strategy. Increasing entropy can give the marginal node a more robust exploration capability and thus better discover potential high-reward policies. α is the temperature parameter, which indicates the relative importance of the entropy term to the cumulative reward.

To calculate the optimal policy π^*, this paper defines the action value function $Q^\pi(s, a)$ and state value function$V^\pi(s)$. $Q^\pi(s, a)$measures the expected cumulative reward obtained by the marginal node based on the caching policy π , considering the entropy of the cached actions, and can be expressed as:

$$Q^{\pi}(s_n,a_n)=E\left[r(s_n,a_n)+\gamma V^{\pi}(s_{n+1})\,|\,s_{n+1}\sim p(s_n,a_n)\right] \quad (9)$$

$V^{\pi}(s)$ measures the expected cumulative reward obtained by the marginal node based on the caching policy π after executing the cached actions, considering the entropy of the cached actions, and can be expressed as:

$$V^{\pi}(s_n)=\pi(s_n)\left[Q^{\pi}(s_n,a)-\alpha\log\pi(a|s_n)|a\sim\pi\right] \quad (10)$$

Two independent neural networks are used to approximate the value function $Q^{\pi}(s,a)$ and the policy function $\pi(\cdot\,|s)$, denoted as $Q_{\omega}(s,a)$ and $\pi_{\theta}(a|s)$, respectively, where ω and θ are the parameters of the two networks. The policy network $\pi_{\theta}(a|s)$ outputs a C-dimensional vector, representing the probability distribution of each cached action. The value network $Q_{\omega}(s,a)$ outputs a score for the cached action. The convergence objective is achieved through stochastic gradient descent during policy evaluation and improvement.

To mitigate the overestimation caused by bootstrapping during the neural network training process, this strategy employs two value networks, $Q_{\omega_1}(s,a)$ and $Q_{\omega_2}(s,a)$, and defines two target value networks, $Q_{\varpi_1}(s,a)$ and $Q_{\varpi_2}(s,a)$, with the same structure as the former but with different parameters. To compute the TD target $\hat{y}_n$, the minimum value between the two target value networks is taken and denoted as:

$$\begin{gathered}\hat{y}_n=r\\ +\gamma\left(min_{i=1,2}Q_{\varpi_i}(s_n,a_n)-\alpha\log\pi_{\theta}(a_n|s_n)\right),\\ a_n\sim\pi_{\theta}(\bullet|s_n)\end{gathered} \quad (11)$$

The TD error, defined as δ, can be represented as the difference between the value network and the TD target, as shown below:

$$\delta_n^i=Q_{\omega_i}(s_n,a_n)-\hat{y}_n, i=1,2 \quad (12)$$

The workflow of the TD-MEAC caching strategy is illustrated in Fig. 4 and described as follows:

- Given the current state s_n, the edge node uses the policy network $\pi_{\theta}(\cdot\,|s_n)$ to obtain a cached action a_n. The agent executes the action, and the environment provides the reward $r(s_n,a_n)$ and the new state s_{n+1}. The transition$(s_n,a_n,r(s_n,a_n),s_{n+1})$ is then stored in the experience buffer D.
- A batch of transitions T is randomly sampled from the experience buffer D, and the TD error of each transition is computed.
- The value and policy network parameters are updated. When updating the parameters of the value network, the objective function is defined as follows:

$$\begin{gathered}J(Q_{\omega})=\\ E\left[\frac{1}{2}\left(Q_{\omega}(s_n,a_n)-\left(r(s_n,a_n)+\gamma\mathbb{E}\left[V^{\varpi}(s_{n+1})\right]\right)\right)^2\right]\end{gathered} \quad (13)$$

Here, $V^{\varpi}(s_{n+1})$ represents the expected action distribution obtained by sampling experiences from the experience buffer using the target value network and (9). When updating the parameters of the policy network, the objective function is defined as follows:

$$J(\pi_{\theta})=E\begin{bmatrix}\alpha\log(\pi_{\theta}(a\,|\,s_n))-\\ min_{i=1,2}Q_{\omega_i}(s_n,a)\,|\,s_n\sim T,a\sim\pi_{\theta}\end{bmatrix} \quad (14)$$

The value network parameters ω_1 and ω_2 are updated using the stochastic gradient descent method based on (9) and (10). Then, the target value network parameters ϖ_1 and ϖ_2 are updated using the weight-based method.

Different temperature parameters must be used depending on the exploration stage. When the policy has essentially completed the exploration of a region and the optimal action has been determined, α should be reduced. Conversely, when the procedure begins exploring a new area, α should be increased to explore more space in search of the optimal action. Consequently, during policy updates, the temperature parameter α is updated synchronously with the constraint formula proposed in [16] to adjust automatically, as shown in (15):

$$J(\alpha)=E\left[-\alpha\left(\log\pi_n(a_n|s_n)+\bar{H}\right)|a_n\sim\pi_n\right] \quad (15)$$

Here, $\bar{H}$ is a constant vector that represents the hyperparameter of the target entropy.

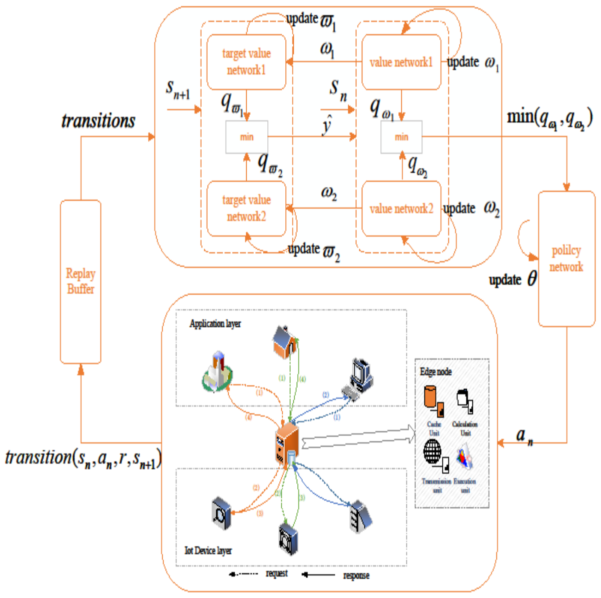

Fig. 4 Workflow of TD-MEAC

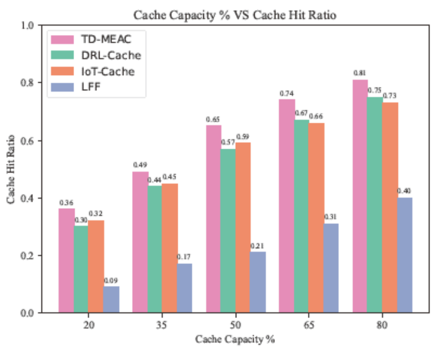

Fig. 5 Cache capacity VS Cache hit ratio

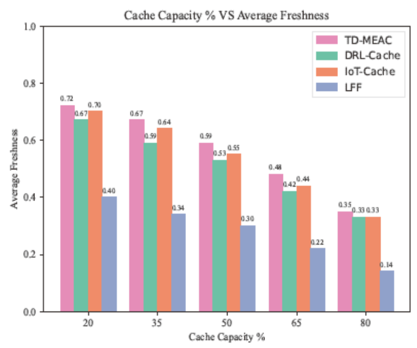

Fig. 6 Cache capacity VS Average Freshness

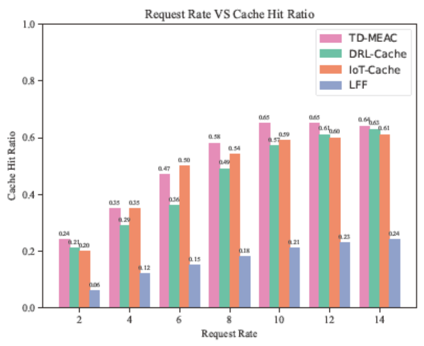

Fig. 7 Request rate VS Cache hit ratio

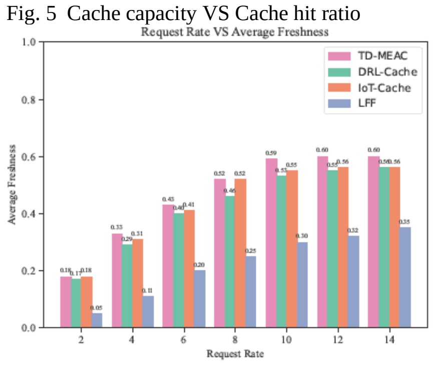

Fig. 8 Request rate VS Average freshness

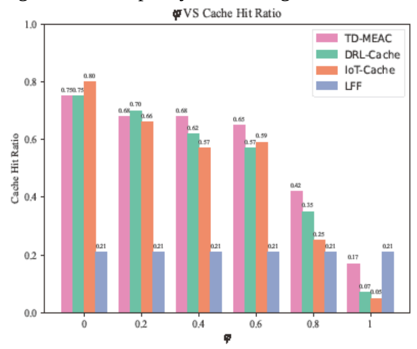

Fig. 9 Weight factor VS Cache hit ratio

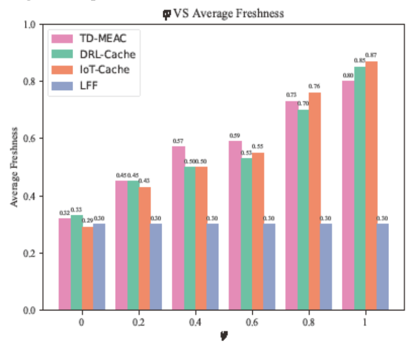

Fig. 10 Weight factor VS Average freshness

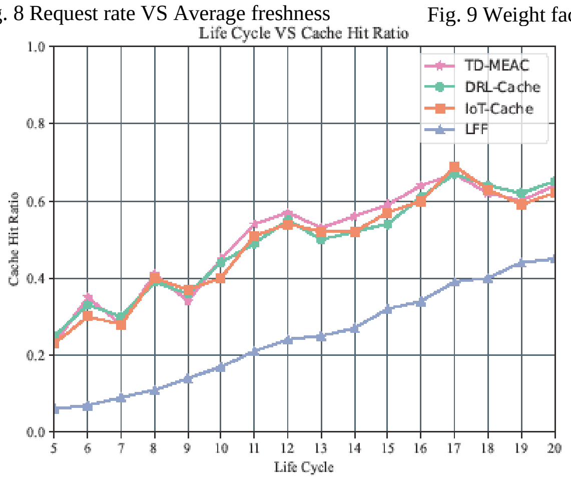

Fig. 11 Life cycle VS Cache hit ratio

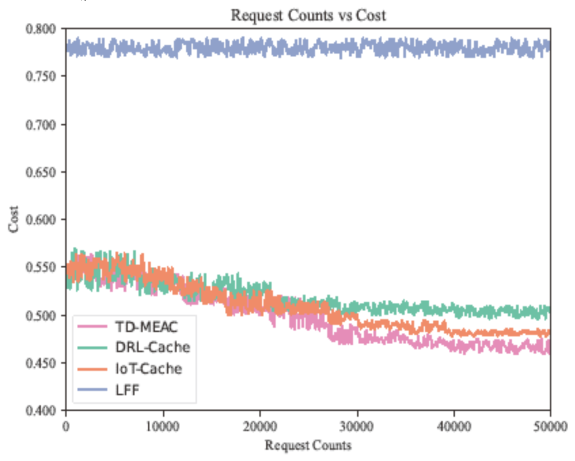

Fig. 12 Request counts VS Cost

IV. Experiments and Evaluation

A. Simulation Setting

The simulations were conducted on a workstation with an AMD Ryzen 7 3700 X 8-Core CPU and 32 GB RAM. The edge node's cache space was set to 100 ($C = 100$), with 200 IoT devices within the coverage range. The data item's lifespan was randomly sampled from 5 to 20-time steps, and the delay in network propagation was uniformly distributed between one to three timesteps. Fifty thousand requests were generated using the Zipf distribution with variable parameter ϵ ranging from 0.9 to 1.7.

Regarding the TD-MEAC neural network model, both the value and policy networks were set to two hidden layers. Each hidden layer had 64 neurons, and the *relu* function was the activation function between the layers. The learning rate was initialized at 0.0001, the discount rate γ was set to 0.99, the initial value of the temperature parameter α was 0.2, and the weight coefficient ψ of the cost was initialized to 0.6. Each batch contained 256 experiences during training, and the experience buffer size was set to 5000.

B. Results and Discussions

This paper compared TD-MEAC with three representative caching schemes, including a baseline caching strategy: LFF[7], and two DRL-based methods: DRL-Cache[13] and IoT-Cache[15].

This section analyzes the impact of different parameter indicators on cache strategies from multiple aspects.

1) Cache capacity: Fig. 5 and Fig. 6 illustrate the impact of varying cache capacity on the cache hit ratio and average freshness of cached items. It can be observed that the cache hit ratio increases, and the average freshness decreases with the rise of cache capacity. For transient data, cache misses are likely to occur if the capacity is small, resulting in a low cache hit ratio. And transient data experiences freshness loss once it is cached until invalid. Larger cache capacity results in fewer cache replacements, reducing the average freshness of cached items.

2) Request rate: As depicted in Fig. 7, the cache hit ratio of the cache strategy increases as the request rate rises from 2 to 10 requests/timestep. However, the cache hit ratio will not continue to grow, and when the request rate exceeds ten requests/timestep, the cache hit ratio tends to stabilize. Fig. 8 shows the average freshness of cached data items increases as the request rate increases. Due to the increase in request rate, the number of cache replacements also increases, and invalidated cache data items are more likely to be replaced, thus increasing the average freshness. The average freshness stabilizes when the request rate exceeds ten requests/timestep. Compared with other cache strategies, the proposed TD-MEAC performs better at different request rates.

3) Weight factor ψ: The weight coefficients in the cost function play an essential role in cache strategies. ψ represents the weight assigned to the cost of freshness loss. As shown in Fig. 9, the cache hit ratio based on the DRL-based cache strategy decreases as ψ increases from zero to one. This is because the cache strategy tends to retrieve new data from the IoT devices to avoid the cost of freshness loss associated with using cached data. As ψ increases, the IoT devices generate more data, decreasing the cache hit ratio. Fig. 10 presents a clear trend of the average freshness. As the proportion of fresh data retrieved from the IoT devices increases, the edge nodes replace cached data items with fresh ones of the same *CID*, resulting in an overall increase in the average freshness.

To investigate the inherent logic of different strategies in the transient data caching process, we recorded the response process of edge nodes to requests. We classified the requested data items according to their lifecycles. The cache hit ratio is shown in Fig. 11. Data items with longer lifecycles tend to have a higher cache hit ratio. For data items with lifecycles between 10 and 16, the cache hit ratio of the TD-MEAC strategy is superior to that of DRL-Cache and IoT-Cache. As for data items with other lifecycles, the performance of the three DRL-based strategies is comparable.

Finally, we recorded the long-term average cost as the number of requests changed. As shown in Fig. 12, the TD-MEAC strategy's average cost converges to the lowest point at around 28,000 requests and maintains relatively stable performance afterward. TD-MEAC outperforms DRL-Cache and IoT-Cache strategies in reducing long-term average costs.

The performance of the proposed TD-MEAC strategy is still better than that of the DRL-Cache and IoT-Cache strategies. Part of the reason is that the TD-MEAC uses the method of entropy regularization and the optimal strategy π^* used contains the entropy of the cache actions. Entropy regularization can encourage the model to balance the predicted distribution of outputs and improve the convergence speed of the model. In addition, the TD-MEAC uses two independent value networks to reduce overestimation problems, making the learning process more stable.

V. Conclusion

This paper addressed the issue of caching transient IoT data in edge networks. A freshness model was established based on the transient data's lifespan and latency, and a cost function was proposed that integrated data freshness and retrieval Cost. To make the approach suitable for dynamic edge environments, we developed a maximum entropy Actor-Critic-based caching strategy, TD-MEAC, reducing reliance on prior knowledge. The performance of TD-MEAC was evaluated experimentally and compared to other strategies. Experimental results indicate that TD-MEAC is better than other caching strategies regarding cache hit ratio, data freshness, and reducing long-term cache costs.

VI. Acknowledgments

The work described in this paper is supported by the National Natural Science Foundation of China (No. 62162003) and the Nanning Science and Technology project (No. 20221031).

References

[1] Al-Ward H, Tan C K, Lim W H. Caching transient data in Information-Centric Internet-of-Things (IC-IoT) networks: A survey[J]. Journal of Network and Computer Applications, 2022: 103491.

[2] Wu H, Fan Y, Wang Y, et al. A comprehensive review on edge caching from the perspective of total process: Placement, policy and delivery[J]. Sensors, 2021, 21(15): 5033.

[3] Maniotis P, Thomas N. Viewport-Aware Deep Reinforcement Learning Approach for 360° Video Caching[J]. IEEE Transactions on Multimedia, 2021.

[4] Zhang Y, Feng B, Quan W, et al. Cooperative edge caching: A multi-agent deep learning based approach [J]. IEEE Access, 2020, 8: 133212-133224.

[5] Wang Y, Friderikos V. Network Orchestration in Mobile Networks via a Synergy of Model-driven and AI-based Techniques [C]//2020 IEEE Eighth International Conference on Communications and Networking (ComNet). IEEE, 2020: 1-5.

[6] Zhong C, Gursoy M C, Velipasalar S. Deep reinforcement learning-based edge caching in wireless networks [J]. IEEE Transactions on Cognitive Communications and Networking, 2020, 6(1): 48-61.

[7] Meddeb M, Dhraief A, Belghith A, et al. Least fresh first cache replacement policy for NDN-based IoT networks [J]. Pervasive and Mobile Computing, 2019, 52: 60-70.

[8] Fatale S, Prakash R S, Moharir S. Caching policies for transient data [J]. IEEE Transactions on Communications, 2020, 68(7): 4411-4422.

[9] Zhang S, Luo H, Li J, et al. Hierarchical soft slicing to meet multi-dimensional QoS demand in cache-enabled vehicular networks [J]. IEEE Transactions on Wireless Communications, 2020, 19(3): 2150-2162.

[10] S. Vural, N. Wang, P. Navaratnam, and R. Tafazolli, "Caching transient data in Internet content routers," IEEE/ACM Trans. Netw., vol. 25, no. 2, pp. 1048-1061, Apr. 2017.

[11] Z. Zhang, C.-H. Lung, I. Lambadaris, and M. St-Hilaire, "IoT data lifetime-based cooperative caching scheme fo r ICN-IoT networks, " in Proc. IEEE ICC, Kansas, MO, USA, May 2018, pp. 1-7.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529-533, 2015.

[13] Zhu H, Cao Y, Wei X, et al. Caching transient data for Internet of Things: a deep reinforcement learning approach [J]. IEEE Internet of Things Journal, 2018, 6(2): 2074-2083.

[14] Wu H, Nasehzadeh A, Wang P. A Deep Reinforcement Learning-Based Caching Strategy for IoT Networks With Transient Data [J]. IEEE Transactions on Vehicular Technology, 2022, 71(12):13310-13319.

[15] Sharma S, Peddoju S K. IoT-Cache: Caching Transient Data at the IoT Edge [C]//2022 IEEE 47th Conference on Local Computer Networks (LCN). IEEE, 2022: 307-310.

[16] Haarnoja T, Zhou A, Hartikainen K, et al. Soft actor-critic algorithms and applications [J]. arXiv preprint arXiv:1812.05905, 2018.