This item was submitted to [Loughborough's Research Repository](#) by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

Contextual learning for content caching with unknown time-varying popularity profiles via incremental clustering

PLEASE CITE THE PUBLISHED VERSION

<https://doi.org/10.1109/tcomm.2021.3059305>

PUBLISHER

Institute of Electrical and Electronics Engineers (IEEE)

VERSION

AM (Accepted Manuscript)

PUBLISHER STATEMENT

Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

LICENCE

All Rights Reserved

REPOSITORY RECORD

Liu, Xingchi, Mahsa Derakhshani, and Sangarapillai Lambotharan. 2021. "Contextual Learning for Content Caching with Unknown Time-varying Popularity Profiles via Incremental Clustering". Loughborough University. <https://hdl.handle.net/2134/14013584.v1>.

Contextual Learning for Content Caching with Unknown Time-Varying Popularity Profiles via Incremental Clustering

Xingchi Liu, Mahsa Derakhshani, *Senior Member, IEEE*, and Sangarapillai Lambotharan, *Senior Member, IEEE*

Abstract—With the rapid development of social networks and high-quality video sharing services, the demand for delivering large quantity and high quality contents under stringent end-to-end delay requirement is increasing. To meet this demand, we study the content caching problem modelled as a Markov decision process in the network edge server when the popularity profiles are unknown and time-varying. In order to adapt to the changing trends of content popularity, a context-aware popularity learning algorithm is proposed. We prove that the learning error of this scheme is sublinear in the number of requests. In light of the learned popularities, a reinforcement learning-based caching scheme is designed on top of the state-action-reward-state-action algorithm with a function approximation. A reactive caching algorithm is also proposed to reduce the complexity. The time complexities of both the caching schemes are studied to demonstrate their feasibility in real time systems and a theoretical analysis is performed to prove that the cache hit rate of the reactive caching algorithm asymptotically converges to the optimal cache hit rate. Finally the simulations are presented to demonstrate the superiority of the proposed algorithms.

Index Terms—Reinforcement learning, Content caching, Context-aware learning, Markov decision process

I. INTRODUCTION

With the proliferation of machine-type communications and increasing user demand for video streaming, content caching is emerging as a key technology to reduce delays experienced by users and network congestion. For instance, according to [1], Internet video is expected to account for 82 percent of all business Internet traffic by 2022. The popularity plays an important role in video traffic, for instance only one percent of the popular Facebook videos represents 83 percent of total watch time [2]. Hence, a proper understanding of the content popularity can help efficient cache deployment and algorithm design.

However, understanding and tracking the variation of the content popularities is challenging. Classical schemes such as the least frequently used (LFU) and the least recently used (LRU) are widely applied in the current caching systems. The LFU can be treated as optimal under an independent reference model (IRM) whereas the LRU reaches the optimal

competitive ratio when the requests are made adversarially. However, they have limitations to handle non-stationary popularity profiles, as it is the case in practical caching networks. In addition, although it is possible to estimate the global popularity in systems such as the on-demand video streaming, the global popularity may not match the local popularity because the servers locating at the network edge can only serve a small geographical area with limited requests, and the variation tendency of the global popularity and the local popularity can be totally different. These challenges motivate us to study the content caching scheme of an edge server under non-stationary and time-varying popularity profiles without any global information.

There are two types of study for content popularities. First, a large proportion of papers employ the IRM to track content popularities and synthesize user request pattern, which assumes the user requests are made in an i.i.d. fashion from a predefined distribution [3]–[5]. On the other hand, there are many papers working on learning the content popularity profiles and then optimizing the caching system. The social networks and the user mobility information are considered to learn the content popularities and various types of machine learning schemes are utilized. However, most of them assume that the prior knowledge of the content popularities is known for the caching placement. Alternatively, a training session is required to learn the file popularities (which means these schemes are not online). These schemes need considerable time and could incur an economic cost to acquire the knowledge of popularity, otherwise it can be outdated and deem to be invalid with non-stationary content popularity.

A. Related Works

Content caching can be applied in many different types of communication systems from cellular networks (e.g., [6]–[8]) to device-to-device networks (e.g., [9], [10]). Many researchers focus on investigating content caching styles including deterministic, probabilistic and coded caching. In [11], [12], deterministic caching schemes were proposed to determine content placement by solving optimization problems according to some knowledge of network and contents. Probabilistic caching (e.g., [13], [14]) assumes that each cache-enabled node has a probability mass function (PMF) for caching different content files rather than deterministically caching the popular files. Coded caching (e.g., [15], [16]) is another caching technology in which each content is partitioned into several coded fragments and each cache-enabled node is only

X. Liu, M. Derakhshani and S. Lambotharan are with Signal Processing and Networks Research Group, Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, U.K (email: X.Liu3@lboro.ac.uk; M.Derakhshani@lboro.ac.uk; S.Lambotharan@lboro.ac.uk).

This work was supported by the Royal Academy of Engineering under the Leverhulme Trust Research Fellowship scheme (Derakhshani-LTRF1920\16\67) and the Engineering and Physical Sciences Research Council (EPSRC) EP/R006385/1.

able to cache part of the fragments so that a request should be served by a set of nodes concurrently.

From another perspective, combining learning and content caching has become a promising research orientation because learning algorithms can be used to estimate file popularity and user request pattern especially with the help of some extra side information. In [6], an extreme-learning machine neural network algorithm was proposed for content popularity prediction by utilizing the learned popularity to initialize the caching capacity of each BS and to decide which files should be cached at each base station. Authors in [17] collected not only the real request information from wireless access points and BSs but also the user mobility information and location information to design a geo-collaborative caching strategy. However, the main drawback of these learning algorithms is the need for prior knowledge and offline training. For these schemes, the collected data for learning could become outdated and the caching performance would be unpredictable.

To address these issues, context-aware online learning has been explored as a new strategy to enhance content caching performance and accelerate learning. In such schemes, context information of requests are collected to help learning. For example, in [18], each video instance with context information was mapped to a point in a context space, and, by grouping different points, videos with similar forecasted popularities maintained an average value for video popularity forecast in future. Moreover, in [19] and [20], caching decision problem is modelled as a contextual multi-arm bandit problem and a reinforcement learning (RL) algorithm is proposed to choose the file that should be cached. Furthermore, [21] and [22] utilized context information by designing a grid-based partitioning method to group requests into different hypercubes and estimate the file popularities. The main drawback of this approach is that the partitioning method is not sufficiently flexible. To improve the clustering process and to learn file popularities better, we propose a context-aware popularity learning to group points into adaptive clusters based on the Euclidean distance via incremental clustering. Numerical results prove that this scheme can create a proper partition to enhance the cache hit rate (CHR).

In addition, RL has been considered as a good learning tool in caching algorithm designs due to its ability to solve decision-making problems in an interactive environment rather than using only a fixed dataset. In [23], a Q-learning based caching policy was proposed to dynamically cache files at small base stations aiming at maximizing the CHR. The work in [24] leveraged RL to perceive the file popularities and to solve a proactive caching problem in wireless networks to minimize the average energy cost. In [25], a model-free RL algorithm was proposed to solve the caching problem for energy harvesting access point. In [26], a MAB problem was formulated to learn the file popularities and to maximize the caching reward.

B. Contributions

The problems identified in both the classical and novel caching schemes reemphasize a practical need for a robust online caching scheme to provide stable caching performance

under time-varying content popularity. In this work, we model a realistic caching decision problem in an edge server as a Markov decision process (MDP) and strictly demonstrate how to learn the popularities of contents online and use the learning results to improve cache management (caching decision). We first design a context-aware popularity learning algorithm to track the time-varying file popularity which is then used in RL to solve the MDP for dynamically updating cached contents in edge servers. Moreover, a reactive caching algorithm is studied to lower the complexity of the RL-based caching scheme. Our algorithm requires neither any prior knowledge about the users and their content requests nor any training sessions, which may be inaccurate, outdated and expensive to obtain. The contributions of this paper are summarized below:

- We propose a context-aware popularity learning algorithm to learn the time-varying file popularity profiles. Particularly, an incremental clustering algorithm is applied to requests with varying context information. This enables us to obtain the similarity among requests, which enhances the learning rate and accuracy.
- The caching decision problem is modelled as a non-stationary MDP. By invoking a linear function approximation, an RL-based content caching scheme is designed via state-action-reward-state-action (SARSA). By incorporating the knowledge learned from the context-aware popularity learning algorithm, the RL is accelerated and the caching decisions are improved. Enlightened by the RL-based caching, a reactive caching algorithm is proposed to reduce the computational complexity.
- A rigorous theoretical analysis on the popularity learning performance is provided and the sublinear learning error over time is demonstrated. We prove that the proposed reactive caching scheme converges to the optimal caching scheme with increasing number of requests and for a given true file popularity. Moreover, the time complexity of the proposed algorithms is shown to be competitively low, which enhances the scope of the algorithms for practical applications.
- Multiple settings of time-varying popularity profiles are designed for performance evaluation, by simulating both the independent and temporal correlated file request processes. Numerical results confirm that both algorithms provide a more robust caching performance as compared to several solutions when the file popularities are unknown and time-varying.

The remainder of this paper is organized as follows. Section II describes the system model. The context-aware popularity learning algorithm is presented in Section III. The detailed RL-based caching algorithm and the reactive caching algorithm are described in Section IV, followed by their theoretical performance in Section V. The simulation setup and results are presented in Section VI, and the conclusions are drawn in Section VII.

II. SYSTEM MODEL

We consider a content delivery network where a content provider (CP) has a library of files, i.e., $\mathcal{F} = \{1, 2, \dots, F\}$. The number of files F may be very large and caching is

a viable solution to improve quality of service. We assume a cache-enabled server at the network edge, the caching capacity of which is M representing the maximum number of contents that can be locally stored. We focus on caching decision problem for a single server to design a decentralized caching scheme and to maximize the CHR of each node independently. Without loss of generality and for the sake of simplicity, we assume the file sizes are identical. Consider req_n as the n^{th} content request made by the users, where $n \in \mathcal{N} = \{1, 2, \dots, N\}$. Each request is represented by a 3-tuple as $\text{req}_n = \langle f_n, t_n, \nu_n \rangle$, where $f_n \in \mathcal{F}$ is the file being requested, t_n is the time that the request was made, and ν_n is the context vector associated with the request. Generally, the context data is a d -dimensional vector and each element represents one context information.

In this section, we model the caching decision process as a non-stationary MDP, which is an appropriate mathematical framework to model sequential decision making considering the random dynamics of the system under study [27]. MDP formulation is useful to model and study the long-term effect of each caching decision on the CHR. In this problem, the server is an agent who decides whether and how to cache the requested file to maximize the long-term CHR. In particular, at each time step $n \in \{1, 2, \dots\}$, the server picks a caching action a_n from the action space $\mathcal{A}$ given the current state of the system $g_n \in \mathcal{G}$, which is the state space. Given the current action and state pair (g_n, a_n) , the server moves to some state g' with the probability of $\Pr(g'|g_n, a_n)$ and receives a reward $r_n(g_n, a_n)$ [28]. In the following, the action space, states, transition probabilities, and reward function are defined.

A. Action Space

When a request arrives, the server checks whether it can be served locally. No change is required if the file is available in the cache. Otherwise, the requested file will be retrieved from CP, and the server decides whether and how to update its cache taking into account the possibility of future requests. Since we assume a time-varying popularity profile in this work, replacing the least popular file in the cache with the requested file is not always optimal. In such a case, it is imperative to take precautions and explore all caching possibilities to some extent to balance the exploitation-exploration trade-off and track time-varying popularity profile. Therefore, the server should be able to cache the requested file by replacing any of the files currently in the cache. Given M is the cache capacity, this implies that the action space of the server has $M + 2$ possible actions as $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_{M+2}\}$ where

- α_1 : The server caches the requested file by replacing the 1st popular file in the cache.
- $\vdots$
- α_M : The server caches the requested file by replacing the M^{th} popular file in the cache.
- α_{M+1} : The requested file is retrieved from CP but not cached.
- α_{M+2} : The server serves the requested file locally without changing its cache.

The server chooses between the first $M + 1$ actions if

the requested file is not cached currently. Otherwise, if the requested file is locally available, the last action α_{M+2} is invoked.

B. State Space and Transition Probability

The state of the process is determined by the currently cached files and the requested file. In particular, the state at time-step n is defined as $g_n = \langle \mathcal{S}_n, f_n \rangle$, where $\mathcal{S}_n$ represents the set of cached files and f_n represents the requested file at time-step n . More specifically, suppose the set of cached files is $\mathcal{S}_n = \{s_n^1, s_n^2, \dots, s_n^M\}$, in which the cached files are indexed in a decreasing order of the file popularities and $\mathcal{S}'$ as the subsequent cache state after taking an action. Moreover, define $f' \in \mathcal{F}$ as the next requested file.

The transitions among the states can be modelled by a Markov chain. Let us define the transition probability from state g_n to state g' . When an action is taken, the cache state transfers from $\mathcal{S}_n$ to $\mathcal{S}'$ deterministically. Therefore, the transition probability depends solely on the file popularity of the next requested file f' at time step n , which is defined as $P_{f'}(t_n)$.

If the requested file $f_n \notin \mathcal{S}_n$ (i.e., cannot be locally served) and $\alpha_m \in \{\alpha_1, \alpha_2, \dots, \alpha_M\}$, the transition probability can be written as

$$\Pr(g'|g_n, a_n = \alpha_m) = \begin{cases} P_{f'}(t_n), & \text{if } g' = \langle s_n^1, \dots, s_n^{m-1}, f_n, s_n^{m+1}, \dots, s_n^M, f' \rangle, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

If the requested file $f_n \notin \mathcal{S}_n$ and action α_{M+1} is taken, the requested file will be served by the CP without replacing the cached files, namely $\mathcal{S}' = \mathcal{S}_n$. In this case, the transition probability is

$$\Pr(g'|g_n, a_n = \alpha_{M+1}) = \begin{cases} P_{f'}(t_n), & \text{if } g' = \langle s_n^1, \dots, s_n^M, f' \rangle, \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Otherwise, if $f_n \in \mathcal{S}_n$, the requested file can be served locally and α_{M+2} will be taken. In this case, the cached files will not be replaced and action and the transition probability is

$$\Pr(g'|g_n, a_n = \alpha_{M+2}) = \begin{cases} P_{f'}(t_n), & \text{if } g' = \langle s_n^1, \dots, s_n^M, f' \rangle, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

The sizes of the state space and the transition probability matrix depend on the file library size and the cache size of the server. In this regard, the size of the state space is $F \times \binom{F}{M}$ and the size of the transition probability matrix is $F^2 \times \binom{F}{M} \times \binom{F}{M}$.

C. Reward Function

In MDP, the reward is the return of the process after transferring from one state to another under a taken action. The CHR is an appropriate metric to evaluate the performance of content caching schemes. The CHR H can be calculated as $H = N_L / N_{\text{Total}}$, where N_L is the number of locally served requests and N_{Total} is the total number of received requests.

Caching relatively popular files can increase the CHR by locally serving more requests. Therefore, we consider the file

popularity when defining the reward. To prioritize caching relatively popular files, the reward function is defined as

$$r_n(g_n, a_n) = \begin{cases} P_{f_n}(t_n) - P_{s_n^m}(t_n), & \text{if } a_n = \alpha_m, \forall m \in \{1, 2, \dots, M\}, \\ 0, & \text{if } a_n = \alpha_{M+1}, \\ P_{f_n}(t_n), & \text{if } a_n = \alpha_{M+2}, \end{cases} \quad (4)$$

where $P_{f_n}(t_n)$ represents the popularity of the requested file and $P_{s_n^m}(t_n)$ represents the popularity of the replaced file. If a cached file is replaced by the requested file, the reward reflects the change of the file popularity as $P_{f_n}(t_n) - P_{s_n^m}(t_n)$. It should be noticed that in this case, the reward can be negative if the requested file is less popular than the replaced file. In addition, if the action α_{M+1} is taken, the cache remains unchanged and hence the reward is set to zero since the requested file is not locally served. Finally, if a requested file is locally served, which refers to the action α_{M+2} , the popularity of the requested file is used as the reward.

D. Action-Value Function

The action-value function is used to quantify how beneficial it is for the agent to perform a given action in a given state under a specific policy. Define $\pi_n(g) = a$ as the policy of choosing the action a under the state g at decision time step n . The action-value function of taking action a in state g at time t under a policy π_n can be defined as

$$Q_n^\pi(g, a) = \mathbb{E}_\pi \left[\sum_{i=n}^N \gamma^{i-n} r_i(g_i, \pi_n(g_i)) | g_n = g, a_n = a \right], \quad (5)$$

where γ is the discount factor showing the importance of the long-term reward compared to the current reward.

III. CONTEXT-AWARE POPULARITY LEARNING

In the formulated MDP in the previous section, the reward function and transition probabilities depend on file popularities. Therefore, first, we need to learn and track popularities in order to solve the MDP. To learn the popularity of files, a simple and effective way is to calculate the frequency of the files that are being requested based on the request history. However, this method faces several problems. First, since the server is located at the network edge, the number of requests is limited in the early stage, thus calculating the frequency might be inaccurate especially when the file popularities are non-stationary and time-varying. In addition, the similarity between files are not considered while multiple files with similar context information may tend to yield close popularities. Therefore, in this section, we propose a context-aware popularity learning algorithm to track the time-varying content popularities.

A. Context Information Management

Context information represents the feature of the content and captures the situation under which the request is made. File request history can be treated as the context information [22]. For example, the context information may include the request counts of a file in the last hour, the last day or the last

Figure 1: Context space

week. With context information, each request req_n is associated with a multi-dimensional context vector ν_n (depends on how many types of context information are available), which can be mapped as a request point in the context space $\mathcal{V}$, where each context information is treated as one axis.

In order to overcome the two deficiencies of popularity learning which were mentioned above, we apply clustering algorithm to the request points to group them into multiple sets. After the clustering process, the request points with the similar context information will be expected to be in the same set. Hence, instead of learning the popularity of each file, we learn the popularity of each set. By utilizing this context information, we will have much more points to accurately estimate the time-varying popularities, which will greatly enhance the accuracy and the learning rate of the popularity estimation.

However, the key challenge is the efficient and dynamic clustering of different request points with a similar context information. K -means clustering is a classic scheme that is able to partition N request points into K clusters. However, K -means is designed only for a fixed number of points, however, in our problem, the number of request points is increasing over time. To solve this problem, inspired by and advancing [29], we propose an incremental clustering-assisted learning algorithm which is suitable for learning the time-varying file popularities by processing the context information incrementally.

Figure 1 illustrates an example of clustering in a two-dimensional context space $\mathcal{V}$. The red points demonstrate various requests received in the course of time, which has been grouped into four clusters. By doing this, the request points with similar context information are grouped into the same cluster and treated to have the same popularity. Since the instant frequency of a requested file can be easily gathered, we consider the average instant frequencies of the request points in a cluster as the popularity of that cluster.

B. Incremental Clustering-Assisted Popularity Learning

Define c_k as the center of cluster $k \in \{1, \dots, K_n\}$, where K_n represents the total number of clusters created till t_n . The

clustering and learning process is divided into three steps as follows.

In the *first* step, once a new request arrives, if K_n is smaller than a certain threshold κ , the server will calculate the Euclidean distances between ν_n (the context point of req_n) and each existing cluster center. Let us define $D_n^{\min}$ as the distance between ν_n and the closest cluster center. If $D_n^{\min}$ is larger than the initial clustering distance threshold δ (which is a predefined parameter), this new context point will be selected as a new cluster center, otherwise, it will be grouped into the closest cluster. The logic behind this step is to create a number of clusters which are not too close to each other for future clustering.

Once a request point is clustered into the existing cluster k , the average popularity $\bar{P}_k$ of cluster k is used as the estimated popularity of this requested file. $\bar{P}_k$ is updated as follows

$$\bar{P}_k = \Sigma_k / \Theta_k, \quad (6)$$

where Σ_k is the sum of the estimated popularity of the points in cluster k and Θ_k is the number of points in cluster k . When a new point is added into cluster k , Σ_k and Θ_k are updated as

$$\Sigma_k = \Sigma_k + q_n, \quad \Theta_k = \Theta_k + 1, \quad (7)$$

where q_n is the real popularity of f_n that can be calculated as the received number of requests for file f_n so far divided by the total number of requests.

In the *second* step, when K_n is equal to κ , the server calculates distances between any two centers. The set of these distances can be represented as $\mathcal{L}$. Define $d_x \in \mathcal{L}$ as the x^{th} smallest distance between any two centers. Then, the sum of the z smallest distances is defined as $\Delta_1 = \sum_{x=1}^z d_x$, where Δ_1 is the clustering distance threshold in phase 1, which will be used to decide whether a new cluster should be created or not once a new request point arrives.

As the number of clusters increases, the clustering distance threshold Δ_r should increase to control the growing rate of the cluster quantity. Specifically, the clustering distance threshold is updated in different phases. Let us define Δ_r as the clustering distance threshold in phase r . Each phase includes the creation of κ clusters. Define l_r as the number of clusters created in phase r . When l_r is larger than κ , it means a sufficient number of clusters has been created based on the current threshold Δ_r . Therefore, Δ_{r+1} will be scaled up to $\xi \Delta_r$ and l_r will be reset to zero. It should be ensured that $\xi > 1$ so that the clustering distance threshold is non-decreasing. By doing this, we can avoid the algorithm from creating too many clusters in a relatively small area in the context space or it may cause clusters overlapping with each other.

In the *third* step, when K_n becomes larger than κ , the proposed algorithm searches for the closest cluster center for an arrived request point and calculates $D_n^{\min}$. Depending on how large $D_n^{\min}$ is compared to the clustering distance threshold Δ_r , the algorithm would decide to map the newly arrived point to the closest cluster or to select it as a new cluster center. This mechanism is implemented stochastically

Algorithm 1 Context-Aware Popularity Learning

Require: $l_r \leftarrow 0, K_n \leftarrow 0, k \leftarrow 1, r \leftarrow 1$

for $n \leftarrow 1$ **to** N **do**

if $K_n < \kappa$ **then**

 Calculate $D_n^{\min}$

if $D_n^{\min} > \delta$ **then**

$c_k \leftarrow \nu_n, K_n \leftarrow K_{n-1} + 1, k \leftarrow k + 1$

else

 Update Σ_k, Θ_k and $\bar{P}_k$ based on (6) and (7)

end if

end if

if $K_n = \kappa$ **then**

 Calculate Δ_1

end if

if $K_n \geq \kappa$ **then**

 Calculate $D_n^{\min}$, generate a Bernoulli binary variable $x_{n,r}$ with $\rho_{n,r} = \min(D_n^{\min} / \Delta_r, 1)$

if $x_{n,r} = 1$ **then**

$c_k \leftarrow \nu_n, K_n \leftarrow K_{n-1} + 1, l_r \leftarrow l_r + 1, k \leftarrow k + 1$

else

 Update Σ_k, Θ_k and $\bar{P}_k$ based on (6) and (7)

end if

if $l_r > \kappa$ **then**

$\Delta_r \leftarrow \xi \Delta_{r-1}, l_r \leftarrow 0, r \leftarrow r + 1$

end if

end if

end for

with the probability defined as $\rho_{n,r} = \min(D_n^{\min} / \Delta_r, 1)$.

More specifically, this new point will be selected as a new cluster center with probability $\rho_{n,r}$, otherwise, it will join the closest cluster with probability $1 - \rho_{n,r}$. Accordingly, if the distance between the new point and the closest cluster center (i.e., $D_n^{\min}$) is much smaller than Δ_r , it is more likely that this point will be added to the closest existing cluster. On the other hand, if $D_n^{\min}$ is comparable with Δ_r , this point would create a new cluster with a higher probability. In particular, if $\rho_{n,r}$ is larger than 1, it means this new point is quite far from existing cluster centers and it will definitely be selected as a new cluster center. The complete context-aware popularity learning algorithm is depicted in Algorithm 1.

IV. CACHING UPDATE ALGORITHMS

We modeled the caching decision process as an MDP in Section II and proposed a learning algorithm to track content popularities in Section III. To solve the caching management MDP, since the transition probabilities are uncertain due to time-varying content popularities, in this section, one of the model-free temporal-difference (TD) learning schemes, namely the SARSA, is utilized to design an optimal caching algorithm [30] without having the dynamics of the environment (transition probabilities).

A. RL-based Caching

In SARSA [31], an agent interacts with the environment and updates the policy based on the taken actions. In particular, SARSA consists of taking an action on a state, noting the reward of the action and the next state, then choosing the next action on the following state, and updating the Q value.

The update of the state-action value (i.e., the Q-value) only depends on the previous Q-value, the reward, and the Q-value of the next state-action; The dynamics of the environment is not needed. After updating the Q-value, the agent moves to the next state and executes the action which has been chosen earlier. The Q-value for a state-action is updated by

$$Q_{n+1}^\pi(g, a) = Q_n^\pi(g, a) + \omega[r_n(g, a) + \gamma Q_n^\pi(g', \pi_n(g')) - Q_n^\pi(g, \pi_n(g))], \quad (8)$$

where ω represents the learning rate which determines to what extent the Q-value is updated based on the newly acquired information and γ is the discount factor.

To decide which action to choose on the current state, the ϵ -greedy policy is introduced with $0 < \epsilon < 1$. According to this policy, the agent either chooses the action which maximizes the Q-function at time step n given the current state with probability $1 - \epsilon$ or randomly chooses an action with probability ϵ . The ϵ -greedy action selection method can be described as

$$\Pr\left(\pi_n(g) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q_n^\pi(g, a)\right) = 1 - \epsilon, \\ \Pr(\pi_n(g) = \operatorname{randi}(\mathcal{A})) = \epsilon. \quad (9)$$

In the TD algorithms such as SARSA, a table is maintained to save all the Q-values of different state-action pairs as the basis of the Q-value update and action selection. However, the size of the state space in our model is very large especially when a large file library and a large cache size are considered, thus directly applying the standard SARSA causes extremely huge memory cost [32]. In addition, this tabular method ignores the correlation between different Q-values, which makes it inefficient to learn the Q-value of each state-action individually, which shows poor generalization quality [33]. As a result, the standard SARSA is difficult and inefficient to be implemented. Therefore, a linear function approximation is utilized to reduce the storage cost and to accelerate the learning due to the fact that the algorithm can generalize its earlier experiences to previously unseen states. The Q-function is represented by a linear combination of a number of features which are able to appropriately reflect the inherent characteristics of the caching system. Therefore, the Q-function can be approximated as

$$Q_n^\pi(g, a) \approx \tilde{Q}_n^\pi(g, a, \theta_n) = \eta_n(g, a)^\top \theta_n, \quad (10)$$

where $\eta_n(g, a)^\top$ is the feature vector and θ_n is the parameter vector at n . $\eta(g, a)_n^\top$ is defined as

$$\eta_n(g, a)^\top = [1, I_n^{\text{req}}(1)P_1(t_n), \dots, I_n^{\text{req}}(F)P_F(t_n), \\ I_n^{\text{cache}}(1)P_1(t_n), \dots, I_n^{\text{cache}}(F)P_F(t_n), \\ I_n^{\text{action}}(1), \dots, I_n^{\text{action}}(M+2)], \quad (11)$$

where I_{req} , I_{cache} , and I_{action} are binary indicators denoting which file is requested, which M files are cached, and which action is taken. Based on the definition, the features of different states and actions are distinguishable, which helps to accurately update the value of the Q-function. To depict the contribution of each feature, the parameter vector $\theta_n \in \mathbb{R}^{2F+M+3}$ is introduced as the weight of each feature. To

Algorithm 2 RL-based caching

Require: γ, ω, ϵ , randomly initialize θ , observe g_0

if the request can be locally served **then**

$a_0 \leftarrow \alpha_{M+2}$

else

Select action a_0 based on (9)

end if

$a_n \leftarrow a_0, g_n \leftarrow g_0$

while $n \leq N$ **do**

Play action a_n , and receive a request req_n

Call Algorithm 1, observe r_n and the next state g_{n+1}

if f_n is cached **then**

$a_{n+1} \leftarrow \alpha_{M+2}$

else

Select the next action a_{n+1} based on (9)

end if

Update θ based on (12),

$g_n \leftarrow g_{n+1}, a_n \leftarrow a_{n+1}, n \leftarrow n + 1$

end while

minimize the function approximation error [34], the gradient descent is applied to update the parameter vector as

$$\theta_{n+1} = \theta_n + \omega[r_n(g, a) + \gamma \tilde{Q}_n^\pi(g', a', \theta_n) - \tilde{Q}_n^\pi(g, a, \theta_n)] \nabla_\theta \tilde{Q}_n^\pi(g, a, \theta_n), \\ = \theta_n + \omega[r_n(g, a) + \gamma \tilde{Q}_n^\pi(g', a', \theta_n) - \tilde{Q}_n^\pi(g, a, \theta_n)] \eta_n(g, a), \quad (12)$$

where the second equation holds since linear approximation is used.

In the RL-based caching, each time a request is received, Algorithm 1 is called to do the clustering and to estimate the file popularity. In the beginning, the cache of the server is assumed to be empty so it always caches the requested files. When the cache is full, the initial state g_0 is observed and the action a_0 is selected based on (9). It is noted that we add a hint to exclude α_{M+2} from the action selection process, thus the server always checks if a request can be locally served to make sure α_{M+2} is always taken if it is possible. After executing the selected action, the algorithm moves to a new state g_{n+1} , and an immediate reward is received. Then the parameter vector is updated by selecting the next action a_{n+1} and following (12). The detailed RL-based caching is described in Algorithm 2.

B. Reactive Caching

Due to the requirement of the real-time processing, the caching decision should be made with the least delay to improve CHR. For the proposed RL-based caching scheme, the computing load still cannot be ignored when a large size of the file library is considered. In addition, in the proposed scheme, there are three parameters that need to be adjusted to reach the optimal performance, which is time-consuming and can cause a generalization problem.

In the RL-based caching scheme, the objective is to maximize the Q-function which is a function of the discounted summation of the file popularities. Solving this problem ensures high CHR since the algorithm tends to cache those popular files. Enlightened by this setting, a reactive caching scheme is designed to overcome the above mentioned problems of

Algorithm 3 Reactive caching

```

for  $n \leftarrow 1$  to  $N$  do
  Receive a request  $\text{req}_n \leftarrow \langle f_n, t_n, \nu_n \rangle$ , call Algorithm
  1, gets  $\tilde{P}_{f_n}$ 
  if The cache is not full then
     $\mathcal{M}_n \leftarrow \mathcal{M}_n \cup f_n$ 
  else
    if  $f_n \in \mathcal{M}_n$  then
       $P_{f_n}(t_n) \leftarrow \tilde{P}_{f_n}(t_n)$ 
    else
      Find  $f_{\text{least}}$ 
      if  $\tilde{P}_{f_n}(t_n) > P_{f_{\text{least}}}(t_n)$  then
         $\mathcal{M}_n \leftarrow \{\mathcal{M}_n \setminus f_{\text{least}}\} \cup f_n, P_{f_n}(t_n) \leftarrow \tilde{P}_{f_n}(t_n)$ 
      end if
    end if
  end if
end for

```

the RL-based caching scheme. In particular, in this scheme, the server will definitely cache the more popular requested file if it cannot be locally served, and drop the least popular file without considering the long-term effect. It should be noted that the reactive caching also can explore the dynamic popularity profiles. The exploration is done by calling the Algorithm 1 which utilizes the context information to track the time-varying file popularity.

To ensure the reactive caching scheme works, the server maintains an extra record of the popularity of the cached files. Each time a request arrives, Algorithm 1 is called to group it into a cluster and estimate the popularity of the requested file, which is defined as $\tilde{P}_{f_n}(t_n)$. If the cache is not full, the server caches the requested file to improve the CHR. Otherwise, for each coming request, the server checks if it can be locally served or not. If this requested file can be locally served, the corresponding popularity of it can be updated by the latest estimated popularity $\tilde{P}_{f_n}(t_n)$. If the requested file cannot be locally served, the popularity of the cached least popular file f_{least} and the requested file f_n are compared, and the server determines to cache the more popular file. The detailed reactive caching scheme is described in Algorithm 3.

V. PERFORMANCE ANALYSIS

In this section, we first bound the learning regret of the file popularity and then utilize it to derive the bound of the CHR of the proposed reactive caching scheme.

A. Learning Regret of File Popularity

There is a widely applied assumption [35] that the expected popularity of files with similar context information is similar. This assumption can be mathematically formulated as follows.

Assumption 1. (Uniform Lipschitz continuity) *There exists a real number $\beta > 0$ such that for any two requests, the popularity difference between the requested files can be bounded as*

$$\mathbb{E}[|q_n - q_{n'}|] \leq \beta \|\nu_n - \nu_{n'}\|, \quad (13)$$

where $\|\cdot\|$ represents the Euclidean norm.

Based on this assumption, we can bound popularity difference of files by the Euclidean norm of the corresponding context points in the context space $\mathcal{V}$. As a result, to bound the learning regret of file popularity, the worst case of the regret should be defined as the largest Euclidean norm between the new point and the farthest point in the same cluster. Before we estimate the total forecast popularity error brought by the first N requests, we prove the following lemma.

Lemma 1. *Let us assume a sequence of N independent experiments denoted by $x_1, x_2, \dots, x_N$, where each experiment succeeds with a probability of $p_n = \min\{\frac{A_n}{B}, 1\}$ where $B \geq 0$ and $A_n \geq 0$ for $n = 1, 2, \dots, N$. If u denotes the random number of consecutive unsuccessful experiments, then with $\mu < 1$, we have*

$$\mathbb{E}\left[\sum_{i=1}^u \left(A_i - \frac{A_i^2}{B}\right)\right] \leq B^\mu, \quad (14)$$

Proof: Let u' be the maximal index for which $p_i < 1$ for all $i \leq u'$. Therefore, we have

$$\begin{aligned} \mathbb{E}\left[\sum_{i=1}^u \left(A_i - \frac{A_i^2}{B}\right)\right] &= \\ \sum_{i=1}^{u'} \left[\left(A_i - \frac{A_i^2}{B}\right) \sum_{i'=i}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right)\right]\right]. \end{aligned} \quad (15)$$

By utilizing mathematical induction, we can rewrite (15) (see Appendix for the proof) as

$$\begin{aligned} \mathbb{E}\left[\sum_{i=1}^u \left(A_i - \frac{A_i^2}{B}\right)\right] &\leq \sum_{i=1}^{u'} \left(A_i - \frac{A_i^2}{B}\right) \prod_{j=1}^i \left(1 - \frac{A_j}{B}\right), \\ &\leq \sum_{i=1}^{u'} \left(A_i - \frac{A_i^2}{B}\right) \prod_{j=1}^{i-1} \left(1 - \frac{A_j}{B}\right). \end{aligned} \quad (16)$$

Let us divide the right-hand side of (16) into two parts and write as

$$\begin{aligned} &\sum_{i=1}^{u'} \left(A_i \prod_{j=1}^{i-1} \left(1 - \frac{A_j}{B}\right) - \frac{A_i^2}{B} \prod_{j=1}^{i-1} \left(1 - \frac{A_j}{B}\right) \right) = \\ &B \sum_{i=1}^{u'} \frac{A_i}{B} \prod_{j=1}^{i-1} \left(1 - \frac{A_j}{B}\right) - \sum_{i=1}^{u'} \frac{A_i^2}{B} \prod_{j=1}^{i-1} \left(1 - \frac{A_j}{B}\right). \end{aligned} \quad (17)$$

Define $q_i = A_i/B < 1$ for $i \leq u'$ as the probabilities of different events to be successful, then (17) can be represented as

$$B \sum_{i=1}^{u'} q_i \prod_{j=1}^{i-1} (1 - q_j) - \sum_{i=1}^{u'} A_i q_i \prod_{j=1}^{i-1} (1 - q_j). \quad (18)$$

It is obvious that $q_i \prod_{j=1}^{i-1} (1 - q_j)$ is the probability that i is the first successful event. Since these events (when i takes different values) are mutually exclusive, $\sum_{i=1}^{u'} q_i \prod_{j=1}^{i-1} (1 - q_j) \leq 1$. Therefore, (18) can be upper bounded as

$$\mathbb{E}\left[\sum_{i=1}^u \left(A_i - \frac{A_i^2}{B}\right)\right] \leq B - \sum_{i=1}^{u'} A_i p_i \prod_{j=1}^{i-1} (1 - p_j). \quad (19)$$

In (19), $\sum_{i=1}^{u'} A_i p_i \prod_{j=1}^{i-1} (1 - p_j)$ can be treated as the expected value of A_i corresponding to the first successful experiment x_i (i.e., $\mathbb{E}[A_i | x_i]$ is the first successful experiment). Obviously, this value is larger than the expected value of A_i no matter x_i succeeds or not. Intuitively, this can be explained as the first succeeding experiment has a larger A_i on average than the average value of A_i . Therefore, the upper bound can be written as

$$\mathbb{E} \left[\sum_{i=1}^u (A_i - A_i^2/B) \right] \leq B - \mathbb{E}[A_i] = B^\mu. \quad (20)$$

Therefore, it can be concluded that there exists an upper bound which is smaller than B (since $\mathbb{E}[A_i] > 0$) and this can be further represented as B^μ with $\mu < 1$.

Lemma 2. *The number of clusters created while serving the first N requests can be upper bounded by $\mathcal{O}(\kappa \log_\xi(vN))$ where $v \geq 1$ is defined as the dataset aspect ratio of the maximum distance to the minimum distance between any two points in the context space.*

Proof: Based on Theorem 3 in [29], consider the phase r' of our algorithm where, for the first time, $\Delta_{r'} \geq \frac{W}{\kappa \log n}$ where n represents the n^{th} point and W is defined as the summation of all distances between each point and its cluster center in the optimal solution. The total number of clusters created in our algorithm before r' can be upper bounded by $\mathcal{O}(\kappa \log_\xi(vn))$. In addition, during and after phase r' can be upper bounded by $\mathcal{O}(\log_\xi n)$. Combining both bounds we can conclude that the number of clusters after N requests is the order of $\mathcal{O}(\kappa \log_\xi(vN))$.

Proposition 1. *The expected total popularity learning error of the first N requests can be upper bounded by $\mathcal{O}(N^\mu)$ for some $\mu < 1$.*

Proof: In the first step of the proposed scheme, when a point is chosen as a cluster center (which means it is the first point in that cluster), it will not cause any error. This is because the caching decision is made only based on the instant request frequency of that file. In the third step, for each coming point, there is a probability $\rho_{n,r}$ for it to be selected as a new cluster center and a probability $1 - \rho_{n,r}$ for it to be grouped into an existing cluster. In the first case of the third step, obviously the forecast popularity error is zero. In the second case, the largest distance (which is the worst-case error) can be defined as $\lambda_n D_n^{\min}$ where λ_n is a factor to scale $D_n^{\min}$ to the worst case (largest distance). Therefore, the expectation of forecast popularity error of one point can be expressed as

$$\mathbb{E}[|\tilde{q}_n - q_n|] \leq \lambda_n D_n^{\min} (1 - \rho_{n,r}) \leq \lambda^{\sup} D_n^{\min} \left(1 - \frac{D_n^{\min}}{\Delta_r} \right), \quad (21)$$

where $\tilde{q}_n$ represents the learned popularity of f_n , q_n represents the file popularity in the optimal scheme, and $\lambda^{\sup}$ is the upper bound of λ_n .

In the phase r , κ clusters are created and this can be interpreted as having κ sequences of unsuccessful experiments if we define a success as creation of a new cluster. For each of these sequences, considering that the upper bound of error

for each experiment is given by (21) and success probability is $\rho_{n,r} = \min(D_n^{\min}/\Delta_r, 1)$, the expected error can be upper bounded by $\lambda^{\sup} (\Delta_r)^\mu$ according to Lemma 1. Consequently, the expected value of the sum of forecast popularity errors in phase r can be upper bounded by $\kappa \lambda^{\sup} (\Delta_r)^\mu$. Mathematically,

$$\mathbb{E} \left[\sum_{n=1}^{n_r} |\tilde{q}_n - q_n| \right] \leq \kappa \lambda^{\sup} (\Delta_r)^\mu, \quad (22)$$

where n_r is the number of requests received in phase r . Therefore, the expected total forecast popularity error of the first N requests is at most

$$\mathbb{E} \left[\sum_{n=1}^N |\tilde{q}_n - q_n| \right] \leq \sum_{r=1}^R \kappa \lambda^{\sup} (\Delta_r)^\mu, \quad (23)$$

where R is the total number of phases corresponding to N requests. Considering that $\Delta_r = \xi^{r-1} \Delta_1$, (23) can be further represented as

$$\mathbb{E} \left[\sum_{n=1}^N |\tilde{q}_n - q_n| \right] \leq \sum_{r=1}^R \kappa \lambda^{\sup} (\xi^{r-1} \Delta_1)^\mu. \quad (24)$$

Since (24) can be treated as the summation of a geometric progression with a common ratio of ξ^μ , the expected total forecast popularity error can be upper bounded by

$$\mathbb{E} \left[\sum_{n=1}^N |\tilde{q}_n - q_n| \right] \leq \frac{\kappa \lambda^{\sup} \Delta_1^\mu ((\xi^\mu)^R - 1)}{\xi^\mu - 1}. \quad (25)$$

Based on Lemma 2, the number of clusters created while serving the first N requests can be upper bounded by $\mathcal{O}(\kappa \log_\xi(vN))$. Therefore, the total number of phases R corresponding to N requests can be upper bounded by

$$R = \mathcal{O}(\kappa \log_\xi(vN)) / \kappa = \mathcal{O}(\log_\xi(vN)). \quad (26)$$

Since the algorithm will step into a new phase when κ cluster centers are created. Consequently, based on (26), (25) can be written as

$$\begin{aligned} \mathbb{E} \left[\sum_{n=1}^N |\tilde{q}_n - q_n| \right] &\leq \frac{\kappa \lambda^{\sup} \Delta_1^\mu ((\xi^\mu)^{\log_\xi(vN)} - 1)}{\xi^\mu - 1}, \\ &= \mathcal{O}((\xi^\mu)^{\log_\xi(vN)}) = \mathcal{O}(N^\mu). \end{aligned} \quad (27)$$

B. Learning Regret of CHR

Similar to [21], let us divide time into periods with each of them containing ϕ requests. Based on the proposed caching scheme, the server will always cache the M -most popular files. In addition, define Q^{sort} as the sorted vector of the popularities of all files in a time period s . For each requested file, assume that the popularity error of file f satisfies

$$|Q_r^{\text{sort}}(f) - Q_e^{\text{sort}}(f)| \leq \Delta Q, \quad f \in \mathcal{F} \quad (28)$$

where $Q_e^{\text{sort}}(f)$ represents the estimated popularities of file f learned by the proposed algorithm and $Q_r^{\text{sort}}(f)$ represents its corresponding real popularity. Based on Proposition 2 from [21], the CHR of the proposed algorithm in time period s can be lower bounded as

$$\tilde{H}_s \geq Q(M) - \frac{2M}{\phi} - \frac{M \cdot \Delta Q_s}{\sum_{f \in \mathcal{F}} Q_r^{\text{sort}}(f)}, \quad (29)$$

where $Q(M) = \frac{\sum_{f=1}^M Q_r^{sort}(f)}{\sum_{f=1}^F Q_r^{sort}(f)}$ represents the normalized total popularity of the M most popular files. This lower bound can be divided into two parts. The first part $(Q(M) - 2M/\phi)$ depends on the caching capacity M and the file popularity distribution (related to $Q(M)$). On the other hand, $\tilde{H}_s$ also depends on the total learning error of file popularities in s (which is denoted as $\Delta Q_s = \sum_{n=1}^{\phi} \Delta Q_n$), since a larger error will lead to a lower CHR. For the optimal scheme, since the full knowledge of file popularity is known, ΔQ_s is set to 0, and the CHR in s is therefore defined as $H_s \geq Q(M) - 2M/\phi$.

Theorem 1. *The CHR of the reactive caching converges to the optimal CHR, namely $\mathbb{E}[\tilde{H}] = \mathbb{E}[H]$.*

Proof: First, we consider the CHR difference between the optimal scheme and the proposed scheme in time period s . The difference can be represented as

$$H_s - \tilde{H}_s \leq \frac{M \cdot \Delta Q_s}{\sum_{f \in \mathcal{F}} Q_r^{sort}(f)} \leq \frac{M \cdot \Delta Q_s}{Q^{inf}}, \quad (30)$$

where Q^{inf} is defined as the lower bound of $\sum_{f \in \mathcal{F}} Q_r^{sort}(f)$.

After that, we wish to calculate the expected difference over all time periods. Therefore, the expected CHR difference can be represented as

$$\begin{aligned} \mathbb{E}[H - \tilde{H}] &\leq \lim_{N \rightarrow \infty} \frac{\sum_{s=1}^{\frac{N}{\phi}} (H_s - \tilde{H}_s)}{\frac{N}{\phi}} = \lim_{N \rightarrow \infty} \frac{\sum_{s=1}^{\frac{N}{\phi}} \frac{M \cdot \Delta Q_s}{Q^{inf}}}{\frac{N}{\phi}}, \\ &= \frac{M}{Q^{inf}} \lim_{N \rightarrow \infty} \frac{\phi}{N} \sum_{s=1}^{\frac{N}{\phi}} \Delta Q_s, \end{aligned} \quad (31)$$

where H and $\tilde{H}$ are the final CHR of the optimal and the proposed caching scheme after receiving N requests. To derive a bound for the mean CHR difference from the first time period to the last, the forecast popularity error can be summed from the first request to the last. Therefore, by considering Proposition 1, the expected difference can be represented as

$$\begin{aligned} \mathbb{E}[H - \tilde{H}] &\leq \frac{M}{Q^{inf}} \lim_{N \rightarrow \infty} \frac{\phi}{N} \sum_{n=1}^N |\tilde{q}_n - q_n|, \\ &\leq \frac{M\phi}{Q^{inf}} \lim_{N \rightarrow \infty} \frac{\mathcal{O}(N^\mu)}{N} = 0. \end{aligned} \quad (32)$$

According to (32), the expected CHR difference between the proposed algorithm and the optimal algorithm is zero, and our algorithm converges to the optimal scheme. Besides, it should be noted that the conclusion holds for any number of file library size.

C. Time Complexity

For the proposed RL-based caching scheme, each time a request arrives, Algorithm 1 is called to estimate the popularity of the requested file, followed by the SARSA algorithm to make a decision with the knowledge of the learned file popularity. In Algorithm 1, we need to find the nearest cluster center of the newly requested file. According to Lemma 2, the number of clusters created while serving n requests can be upper bounded by $\mathcal{O}(\kappa \log_\xi(vn))$. Therefore, the time complexity of finding the nearest cluster center can be represented as $\mathcal{O}(\log_\xi(vn))$. In SARSA, before deciding an action to take, matrix multiplication is needed to calculate

the Q-value of each action. Recall that M denotes the cache size and F denotes the file library size. Since the size of feature vector is $(M + 2, 2F + M + 3)$ and the size of the parameter vector is $(2F + M + 3, 1)$, the time complexity of calculating the multiplication of the matrices is $\mathcal{O}(MF)$. Therefore, the time complexity of the proposed RL-based caching is $\mathcal{O}(\log_\xi(vn) + MF)$.

For the proposed reactive caching scheme, it does not use matrix multiplication but only calls Algorithm 1 and sorts the cached files in a decreasing order of popularities. Therefore, the time complexity can be represented as $\mathcal{O}(\log_\xi(vn)) + M$. Comparing both algorithms, we can find that both algorithms reach low time complexity. However, in practice, there is always a limited cache size but a potentially large file library. In this case, RL-based caching leads to higher time complexity, which highlights the significance of designing the reactive caching scheme.

VI. SIMULATION RESULTS

In this section, the performance of the proposed caching schemes is evaluated and compared with other existing algorithms. The effect of parameters used in the context-aware popularity learning scheme is also investigated.

A. Simulation Setup

The file popularities are modelled using a Zipf distribution which is widely used in content caching literature [7], [10], [36]. The PMF of Zipf distribution is written as $P_f = \frac{f^{-\psi}}{\sum_{f=1}^F f^{-\psi}}$, where P_f is the popularity of f , F is the file library size, and ψ is the skewness factor ($\psi \geq 0$). A larger ψ means the popular file becomes more popular. When $\psi = 0$, the Zipf distribution will be converted to a uniform distribution. In addition, the request arrival is assumed to follow a Poisson process which implies the time interval between the two consecutive requests follows an exponential distribution with a rate parameter ζ which increases for more frequent requests.

The simulation lasts for 72000 time slots. The duration of each time slot is one minute and $F = 100$. To evaluate the robustness of the proposed algorithms against time-varying popularity profiles, a deterministic variation is introduced to the file popularity distribution by randomly permuting all the files at every 12000 time slots. The learning rate, the discount factor and the ϵ -greedy factor are set to 0.1, 0.05 and 0.1, respectively. All the results are generated by running the simulation for five times and calculating the average CHR. In addition, the request history is chosen to be the context information in the simulation. Specifically, we use the total number of requests for a file in the past week and the past day as the context information in the simulation.

As a *benchmark*, we simulate LRU, LFU, Least frequent recently used (LFRU) [37] and a popularity-driven (POP) caching [21] algorithm. For the LRU scheme, the node always tries to update its cache by replacing the least recently requested file with the newly requested file. Besides, LRU is suitable for the case that the popularity of file changes over time because it does not consider the request history when making a decision. In LFU scheme, the node always cache

Figure 2: CHR over time

Figure 4: CHR versus caching capacity

Figure 3: CHR over time

Figure 5: CHR versus ζ

the most frequently requested file. However, the caching performance might drop drastically if the file popularity changes. In addition, LFRU combines benefits of both LRU and LFU by partitioning the cache into two parts. It is a well known scheme for content caching network. Finally, POP caching is also simulated which is a context-aware caching algorithm with the ability to handle the time-varying file popularity.

B. Numerical Results

Figure 2 illustrates the CHR of different caching algorithms over time. The caching capacity is set to 10 files and ψ is set to 1. The proposed algorithms can identify the changes of the popularity profiles. In addition, the RL-based caching scheme reaches the highest CHR and competitive robustness among all six schemes because it introduces a reasonable amount of exploration to deal with the variability of the popularity profiles thus caching the more popular files when the popularity profiles change. Moreover, the reactive caching reaches a competitive CHR and robustness as well. Note that before the variation happens, this scheme performs well because the proposed online learning algorithm is able to quickly learn the popularity profiles and the incremental clustering is more efficient than the grid-based clustering scheme utilized in the POP caching scheme. Finally, for the rest benchmark schemes, it is obvious that LFU can perform well before any variations but its robustness is very poor. LRU is very robust but cannot reach a good CHR when the storage resource is limited. As a combination of LRU and LFU, LFRU performs much better, however, there is an obvious gap because LFRU does not consider the context information. Besides, this scheme is

quite heuristic which means it needs tuning to reach a better performance.

Figure 3 shows the caching performance over time when the period of the popularity variation decreases from 12000 to 6000 time slots. Therefore, the variation happens more frequently and the RL-based caching scheme can reach the highest CHR. In addition, due to the more frequent variations, the gap between the RL-based caching and the reactive caching becomes larger, which further demonstrates the robustness of the proposed RL-based caching scheme. Moreover, the reactive caching scheme still reaches a higher CHR as compared to the benchmark schemes.

Figure 4 depicts the caching performance versus the capacity which ranges from 3 to 20 (files). The CHRs are collected at the end of the simulation with $\psi = 1$. Obviously, the proposed RL-based caching scheme outperforms other benchmark schemes for different cache capacities. Furthermore, the reactive caching scheme reaches a competitive CHR as well which proves that the proposed schemes are suitable and robust for the cache-limited scenario.

In Figure 5, the effect of the request arrival rate is studied by changing the rate parameter ζ of the exponential distribution. The caching capacity here is set to 10 files and ψ is set to 1. The performance of the proposed caching schemes is robust against ζ because according to the figure, the choice of the length of context information is not sensitive to the request arrival rate. To further study the effect of the file library size on the CHR, we simulate a new scenario with 1000 files in the file library and the cache size is set to 20. As we observe in Figure 6, the proposed RL-based caching algorithm reaches the

Figure 6: CHR over time

Figure 8: CHR versus file variation percentage

Figure 7: CHR versus popularity variation percentage

highest CHR and it is very robust against the frequent changes of the file popularity profiles. In addition, the reactive caching scheme also reaches a competitive performance as compared to other benchmark schemes. Finally, the running times of both the proposed caching schemes are measured under this setting. Both the designed algorithms were run on an i5 desktop computer by MATLAB. In order to accurately measure the running time, both schemes are executed for 1000 time slots. The running times for RL-based caching and reactive caching are 1159 seconds and 642 seconds, respectively. According to the measurements, reactive caching runs faster than RL-based caching, which demonstrates that the former scheme requires a lower computational complexity.

In the previous results, the change of the popularity profile is assumed to be independent over time and the popularities of all the files are changed at each variation. Inspired by [36], we further test the caching performance of the proposed algorithms with correlated file popularity, which means only a part of file popularities are changed, so the popularity profile is dependent over time and correlated with previous profile. In Figure 7, the variation percentage of popularity profiles ranges from 60% to 100%. According to the results, with the variation percentage increasing, the CHR of the proposed schemes drop because the popularity profiles are more dynamic. However, the RL-based caching scheme still reaches the best performance and the performance gap between it and the reactive caching becomes larger when the popularities of more files vary. In order to further evaluate the caching performance, a new scenario is simulated. For each variation of popularity profiles, a subset of the files are removed from

Figure 9: CHR over time-Bernoulli request model

the file library and a number of new files are included. Besides, a part of the file popularities are still changed periodically. Figure 8 presents the caching performance for this dynamic scenario. The RL-based scheme reaches the highest CHR, which demonstrates the robustness of it under a correlated time-varying popularity profiles.

C. Correlated File Request Process

In this section, we designed two new settings to extend the simulation results, which capture the temporal correlation of file requests in varying ways. The first one is based on the Bernoulli model [36] and the second is based on the shot noise model (SNM) [38]. To implement the time-varying popularity profile, a part of the popularity profile is changed periodically consistent with previous results. In Figure 9, we study the Bernoulli request model which can capture the request correlation. Assume there are 10 users and each user can make a file request with the probability of 0.1 in each time slot and the file library size is 1000. The result reflects that the proposed algorithms perform well in this new setting and the correlated requests will not affect the caching performance as compared to the results of independent request model (Figure 6).

In addition, we also simulate the SNM which is able to capture the popularity evolution and explicitly account for the temporal locality of file requests. Following the setting in [39], the requests for the contents in a window of time slots are generated based on the SNM using the exponential shape [38]. The average number of requests for each content in the considered time window follows the Zipf distribution. The result is presented in Figure 10. According to the results, we

Figure 10: CHR over time-SNM

Figure 11: CHR versus δ

conclude that both the proposed caching schemes outperform all the benchmarks.

D. Parameter Determination

Here, we investigate the effect of parameters of the proposed context-aware popularity learning algorithm. The popularity profiles are assumed to be unchanged. The optimal caching scheme which is assumed to know the true file popularity is also simulated. For the optimal caching, the server caches the most popular files without any updates.

Figure 11 depicts the CHR and the number of clusters versus δ . When δ is relatively small, the CHR gap between the optimal scheme and the reactive caching scheme is very small. However, when $\delta \geq 100$, the gap starts to increase which means the popularity learning scheme gradually loses its learning accuracy. The reason is that when δ is too large, only very few clusters will be created so that the request points with different popularity levels can be grouped into the same cluster. Therefore, there is a high probability that the algorithm cannot identify the difference between the two requests. In the simulation, to avoid creating too few clusters, δ is set to 5.

Figure 12 demonstrates the caching performance versus κ . It is observed that as κ increases, the CHR difference between the optimal scheme and the reactive scheme first increases and then remains stable. This means that creating too many clusters cannot necessarily help to improve the learning accuracy but may lead to confusion of popularity learning. In our simulations, to avoid the algorithm creating too few clusters and thus reducing the CHR, κ is set to 20. We also investigate the effect of ξ and z . The results show that these parameters do not affect the caching performance severely but only affect the speed of creating clusters. Therefore, ξ and z

Figure 12: CHR versus κ
are chosen as 10 and 15 respectively.

VII. CONCLUSION

In this paper, we studied a caching decision problem in the network edge server under a dynamic and time-varying file popularity profile. We first designed a context-aware popularity learning algorithm to improve the accuracy and speed of tracking the file popularity. With the assistance of this algorithm, an RL-based caching scheme is proposed to make proper caching decisions and to improve the CHR. Furthermore, a reactive caching algorithm was designed to reduce the computational complexity for real time processing. Via the theoretical analysis, we demonstrated the superiority and the efficacy of the proposed algorithms. Finally, through numerical results, we demonstrated that the proposed algorithms are able to achieve a competitive and robust caching performance as compared to various benchmark schemes.

APPENDIX

In order to verify the transition from (15) to (16), we need to prove the following equation

$$\begin{aligned} & \mathbb{E}\left[\sum_{i=1}^u \left(A_i - \frac{A_i^2}{B}\right)\right] \\ &= \sum_{i=1}^{u'} \left[\left(A_i - \frac{A_i^2}{B}\right) \sum_{i'=i}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right)\right]\right], \\ &\leq \sum_{i=1}^{u'} \left(A_i - \frac{A_i^2}{B}\right) \prod_{j=1}^i \left(1 - \frac{A_j}{B}\right). \end{aligned} \quad (33)$$

This can be proved via the mathematical induction. First, let us rewrite (33) and prove,

$$\sum_{i'=i}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right)\right] \leq \prod_{j=1}^i \left(1 - \frac{A_j}{B}\right). \quad (34)$$

For $i = 1$, (34) is equal to $\sum_{i'=1}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right)\right] \leq \left(1 - \frac{A_1}{B}\right)$.

For $i = X$, we assume the following equation holds,

$$\sum_{i'=X}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right)\right] \leq \prod_{j=1}^X \left(1 - \frac{A_j}{B}\right). \quad (35)$$

Then, when $i = X + 1$, we need to prove that

$$\sum_{i'=X+1}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right)\right] \leq \prod_{j=1}^{X+1} \left(1 - \frac{A_j}{B}\right). \quad (36)$$

The left-hand side of (36) can be written as

$$\begin{aligned} & \sum_{i'=X+1}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right) \right] \\ &= \sum_{i'=X}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right) \right] - \frac{A_{X+1}}{B} \prod_{j=1}^X \left(1 - \frac{A_j}{B}\right). \end{aligned} \quad (37)$$

According to the assumption in (36) and (37), we have

$$\begin{aligned} & \sum_{i'=X+1}^{u'} \left[\frac{A_{i'+1}}{B} \prod_{j=1}^{i'} \left(1 - \frac{A_j}{B}\right) \right] \\ & \leq \prod_{j=1}^X \left(1 - \frac{A_j}{B}\right) - \frac{A_{X+1}}{B} \prod_{j=1}^X \left(1 - \frac{A_j}{B}\right), \\ & = \prod_{j=1}^X \left(1 - \frac{A_j}{B}\right) \left(1 - \frac{A_{X+1}}{B}\right) = \prod_{j=1}^{X+1} \left(1 - \frac{A_j}{B}\right). \end{aligned} \quad (38)$$

Therefore, we complete the proof of the transition from (15) to (16).

REFERENCES

- [1] "2020 global networking trends report:," Cisco, Tech. Rep., 2020.
- [2] L. Tang, Q. Huang, A. Puntambekar, Y. Vigfusson, W. Lloyd, and K. Li, "Popularity prediction of facebook videos for higher quality streaming," in *USENIX Annual Technical Conference*, Santa Clara, Jul 2017, pp. 111–123.
- [3] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, "The role of caching in future communication systems and networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1111–1125, Jun 2018.
- [4] H. Kim, J. Park, M. Bennis, S. Kim, and M. Debbah, "Ultra-dense edge caching under spatio-temporal demand and network dynamics," in *IEEE International Conference on Communications (ICC)*, Paris, May 2017.
- [5] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *IEEE INFOCOM*, San Francisco, Apr 2016.
- [6] S. S. Tanzil, W. Hoiles, and V. Krishnamurthy, "Adaptive scheme for caching youtube content in a cellular network: Machine learning approach," *IEEE Access*, vol. 5, pp. 5870–5881, Mar 2017.
- [7] Z. Chen, J. Lee, T. Q. Quek, and M. Kountouris, "Cooperative caching and transmission design in cluster-centric small cell networks," *IEEE Trans. on Wireless Communications*, vol. 16, no. 5, pp. 3401–3415, May 2017.
- [8] X. Zhang, G. Zheng, S. Lambotharan, M. R. Nakhai, and K. Wong, "A reinforcement learning-based user-assisted caching strategy for dynamic content library in small cell networks," *IEEE Trans. on Communications*, vol. 68, no. 6, pp. 3627–3639, 2020.
- [9] C. Ma, M. Ding, H. Chen, Z. Lin, G. Mao, Y.-C. Liang, and B. Vucetic, "Socially aware caching strategy in device-to-device communication networks," *IEEE Trans. on Vehicular Technology*, vol. 67, no. 5, pp. 4615–4629, May 2018.
- [10] D. Malak, M. Al-Shalash, and J. G. Andrews, "Optimizing content caching to maximize the density of successful receptions in device-to-device networking," *IEEE Trans. on Communications*, vol. 64, no. 10, pp. 4365–4380, Oct 2016.
- [11] C. Zhan and Z. Wen, "Content cache placement for scalable video in heterogeneous wireless network," *IEEE Communications Letters*, vol. 21, no. 12, pp. 2714–2717, Dec 2017.
- [12] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1258–1275, Jun 2018.
- [13] K. Li, C. Yang, Z. Chen, and M. Tao, "Optimization and analysis of probabilistic caching in n -tier heterogeneous networks," *IEEE Trans. on Wireless Communications*, vol. 17, no. 2, pp. 1283–1297, Feb 2018.
- [14] X. Xu and M. Tao, "Analysis and optimization of probabilistic caching in multi-antenna small-cell networks," in *IEEE GLOBECOM*, Singapore, 2017.
- [15] —, "Modeling, analysis, and optimization of coded caching in small-cell networks," *IEEE Trans. on Communications*, vol. 65, no. 8, pp. 3415–3428, Aug 2017.
- [16] J. Zhang, X. Lin, and X. Wang, "Coded caching under arbitrary popularity distributions," *IEEE Trans. on Information Theory*, vol. 64, no. 1, pp. 349–366, Jan 2018.
- [17] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, "Understanding performance of edge content caching for mobile video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1076–1089, May 2017.
- [18] J. Xu, M. van der Schaar, J. Liu, and H. Li, "Forecasting popularity of videos using social media," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 2, pp. 330–343, 2015.
- [19] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Smart caching in wireless small cell networks via contextual multi-armed bandits," in *IEEE ICC*, Kuala Lumpur, 2016.
- [20] S. Muller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Trans. on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, Feb 2017.
- [21] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *IEEE INFOCOM*, Berlin, 2016.
- [22] —, "Trend-aware video caching through online learning," *IEEE Trans. on Multimedia*, vol. 18, no. 12, pp. 2503–2516, Dec 2016.
- [23] K. Guo, C. Yang, and T. Liu, "Caching in base station with recommendation via q-learning," in *IEEE WCNC*, San Francisco, 2017.
- [24] S. O. Somuyiwa, A. György, and D. Gündüz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1331–1344, Jun 2018.
- [25] L. Dong, D. Niyato, D. I. Kim, and D. T. Hoang, "A joint scheduling and content caching scheme for energy harvesting access points with multicast," in *IEEE GLOBECOM*, Singapore, 2017.
- [26] J. Song, M. Sheng, T. Q. S. Quek, C. Xu, and X. Wang, "Learning-based content caching and sharing for wireless networks," *IEEE Trans. on Communications*, vol. 65, no. 10, pp. 4309–4324, Oct 2017.
- [27] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- [28] J. Y. Yu and S. Mannor, "Online learning in markov decision processes with arbitrarily changing rewards and transitions," in *2009 International Conference on Game Theory for Networks*, Istanbul, 2009, pp. 314–322.
- [29] E. Liberty, R. Sriharsha, and M. Sviridenko, "An algorithm for online k-means clustering," in *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments*, Arlington, pp. 81–89.
- [30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [31] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, England, 1994, vol. 37.
- [32] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Approximate reinforcement learning: An overview," in *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Paris, 2011.
- [33] N. K. Jong and P. Stone, "Model-based function approximation in reinforcement learning," in *ACM International Joint Conference on Autonomous Agents and Multiagent Systems*, Honolulu, 2007.
- [34] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *ACM International Conference on Machine Learning*, Helsinki, 2008.
- [35] A. Slivkins, "Contextual bandits with similarity information," *Journal of Machine Learning Research*, vol. 15, pp. 2533–2568, 2014.
- [36] B. N. Bharath, K. G. Nagananda, D. Gündüz, and H. V. Poor, "Caching with time-varying popularity profiles: A learning-theoretic perspective," *IEEE Trans. on Communications*, vol. 66, no. 9, pp. 3837–3847, Sep 2018.
- [37] M. Bilal and S. Kang, "A cache management scheme for efficient content eviction and replication in cache networks," *IEEE Access*, vol. 5, pp. 1692–1701, 2017.
- [38] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, "Temporal locality in today's content caching: Why it matters and how to model it," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 5, p. 5–12, Nov. 2013.
- [39] J. Gao, S. Zhang, L. Zhao, and X. S. Shen, "The design of dynamic probabilistic caching with time-varying content popularity," *IEEE Transactions on Mobile Computing*, 2020.

and live video transcoding.

Xingchi Liu received B.Eng. degree in telecommunications engineering from the Xidian University, China, in 2015, and the M.Sc. degree in communication networks and signal processing from the University of Bristol, U.K. in 2017. He was an Intern at Toshiba Europe Limited, Bristol Research & Innovation Laboratory, U.K. He is currently working toward the Ph.D degree with the Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, U.K. His research interests include machine learning, content caching

versity, from 2013 to 2015. She is currently a Senior Lecturer in digital communications and the Royal Academy of Engineering/Leverhulme Trust Research Fellow with the Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, U.K. Her research interests include radio resource management, ultra-reliable low latency communications, applications of machine learning, and optimization for wireless networks. She received several awards and fellowships, including the John Bonsall Porter Prize, the McGill Engineering Doctoral Award, the Fonds de Recherche du Québec–Nature et Technologies (FRQNT), and the Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowships. She serves as the Editor for IEEE WIRELESS COMMUNICATIONS LETTERS and IET Signal Processing Journal.

Mahsa Derakhshani (Senior Member, IEEE) received the Ph.D. degree in electrical engineering degree from McGill University, Montréal, Canada, in 2013. She was an Honorary NSERC Postdoctoral Fellow with the Department of Electrical and Electronic Engineering, Imperial College London, from 2015 to 2016, a Post-Doctoral Research Fellow with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, and a Research Assistant with the Department of Electrical and Computer Engineering, McGill Uni-

UTRAN. He was with King's College London and Cardiff University as a Lecturer and a Senior Lecturer, respectively, from 2002 to 2007. He is currently a Professor of digital communications and the Head of the Signal Processing and Networks Research Group, Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, U.K. His current research interests include 5G networks, MIMO, blockchain, machine learning, and network security. He has authored more than 200 journal articles and conference papers in these areas. He serves as an Associate Editor for IEEE TRANSACTIONS ON SIGNAL PROCESSING.

Sangarapillai Lambotharan (Senior Member, IEEE) received the Ph.D. degree in signal processing from Imperial College London, U.K., in 1997. He was a Visiting Scientist with the Engineering and Theory Centre, Cornell University, USA, in 1996. Until 1999, he was a Post-Doctoral Research Associate with Imperial College London. From 1999 to 2002, he was with the Motorola Applied Research Group, U.K., where he investigated various projects, including physical link layer modeling and performance characterization of GPRS, EGPRS, and