Optimizing Video Caching at the Edge: A Hybrid Multi-Point Process Approach

Xianzhi Zhang, Yipeng Zhou, *Member, IEEE*, Di Wu, *Senior Member, IEEE*, Miao Hu, *Member, IEEE*, Xi Zheng, *Member, IEEE*, Min Chen, *Fellow, IEEE*, and Song Guo, *Fellow, IEEE*

Abstract—It is always a challenging problem to deliver a huge volume of videos over the Internet. To meet the high bandwidth and stringent playback demand, one feasible solution is to cache video contents on edge servers based on predicted video popularity. Traditional caching algorithms (e.g., LRU, LFU) are too simple to capture the dynamics of video popularity, especially long-tailed videos. Recent learning-driven caching algorithms (e.g., DeepCache) show promising performance, however, such black-box approaches are lack of explainability and interpretability. Moreover, the parameter tuning requires a large number of historical records, which are difficult to obtain for videos with low popularity. In this paper, we optimize video caching at the edge using a white-box approach, which is highly efficient and also completely explainable. To accurately capture the evolution of video popularity, we develop a mathematical model called *HRS* model, which is the combination of multiple point processes, including Hawkes' self-exciting, reactive and self-correcting processes. The key advantage of the HRS model is its explainability, and much less number of model parameters. In addition, all its model parameters can be learned automatically through maximizing the Log-likelihood function constructed by past video request events. Next, we further design an online HRS-based video caching algorithm. To verify its effectiveness, we conduct a series of experiments using real video traces collected from Tencent Video, one of the largest online video providers in China. Experiment results demonstrate that our proposed algorithm outperforms the state-of-the-art algorithms, with 15.5% improvement on average in terms of cache hit rate under realistic settings.

Index Terms—Video caching, edge servers, point process, Monte Carlo, gradient descent

1 Introduction

Due to the fast growth of the online video market, the online video streaming service has dominated the Internet traffic. It was forecasted by Cisco [1] that video streaming applications will take up the Internet traffic from 59% in 2017 to 79% in 2022. On one hand, online video providers need to stream HD (high definition) videos with stringent playback requirements. On the other hand, both video population and user population are growing rapidly. Thereby, edge devices have been pervasively exploited by online video providers to cache videos so as to reduce the Internet traffic and improve the user Quality of Experience (QoE)[2], [3], [4], [5], [6], [7].

We consider the video caching problem on edge servers that can provide video streaming services for users in a certain area (e.g., a city). If a video is cached, edge servers can stream the video to users directly with a shorter response time. In contrast, requests for videos that are missed by edge servers have to be directed to remote servers, resulting in lower QoE. From video providers' perspective, the target is to maximize the cache hit rate of user video requests by properly caching videos on edge servers. Different from the general caching problem in [8], [9], [10], [11], video caching on edge servers confront two particular challenges. Firstly, video requests are driven by users' view interest in different locations [2], [7]. Thus, it is crucial to make caching decisions based on future video popularity which can be learnt from localized request events. Secondly, user view interest can be highly dynamic [5], [6]. Thereby, it is essential to design an agile video caching algorithm that can swiftly change video popularity prediction according to user requests.

Briefly, there are two approaches to predict video popularity so as to make video caching decisions. The first

- *Xianzhi Zhang, Di Wu, and Miao Hu are with the Department of Computer Science, Sun Yat-sen University, Guangzhou, Guangdong 510006, China, and also with the Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, Guangdong 510006, China. E-mail: zhangxzh9@mail2.sysu.edu.cn, {wudi27, humiao5}@mail.sysu.edu.cn.*
- *Yipeng Zhou is with the School of Computing, FSE, Macquarie University, Macquarie Park, NSW 2122, Australia, and also with Peng Cheng Laboratory, Shenzhen, Guangdong 518000, China. E-mail: yipeng.zhou@mq.edu.au.*
- *Xi Zheng is with the School of Computing, FSE, Macquarie University, Macquarie Park, NSW 2122, Australia. E-mail: james.zheng@mq.edu.au.*
- *Min Chen is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: minchen@ieee.org.*
- *Song Guo is with the Department of Computing, The Polytechnic University of Hong Kong, Hung Hom, Hong Kong. E-mail: song.guo@polyu.edu.hk.*

Manuscript received 30 Jan. 2021; revised 3 Dec. 2021; accepted 16 Jan. 2022. Date of publication 1 Feb. 2022; date of current version 4 Apr. 2022.
This work was supported in part by the National Natural Science Foundation of China under Grants U1911201, U2001209, 62072486, 61872310 and 62176101, in part by the Science and Technology Planning Project of Guangdong Province under Grant 2021A0505110008, in part by the Australia Research Council under Grant DE180100950, in part by the Major Key Project of PCL under Grant PCL2021A08, in part by the Australian Research Council Linkage Project under Grant LP190100676, in part by the Hong Kong RGC Research Impact Fund (RIF) under Grant R5060-19, in part by the General Research Fund (GRF) under Grants 152221/19E, 152203/20E, and 152244/21E, and in part by Shenzhen Science and Technology Innovation Commission under Grant R2020A045.
(Corresponding author: Di Wu)
Recommended for acceptance by V. Chaudhary.
Digital Object Identifier no. 10.1109/TPDS.2022.3147240

approach is to predict video popularity based on empirical formulas. Classical LRU (Least Recently Used), LFU (Least Frequently Used) and their variants [12], [13], [14] are such video caching algorithms. However, such approach is subject to the difficulty of choosing parameter values. For example, it is not easy to choose an appropriate time window size in LRU and LFU algorithms [12], [15]. The second approach is learning-driven video caching algorithm. Typically, NN (neural network) models such as LSTM (Long Short-Term Memory) [16], [17] can be leveraged to predict video popularity so as to make right video caching decisions. The strength of this approach is that model parameters can be automatically determined through learning historical request patterns, and thus such algorithms can achieve better video caching performance than classical algorithms. Yet, such models require a long training time, and are lack of explainability and interpretability.

In this paper, we aim at optimizing the performance of video caching on edge servers using a white-box approach. To this purpose, we develop a mathematical model called *HRS* model to capture the evolution process of video popularity. The HRS model is the combination of multiple point processes, including the Hawkes process [18], the reactive process [19] and the self-correcting process [20]. The point processes enable us to exploit timestamp information of historical video requests and time-varying kernels to predict future events.

Specifically, the intuition behind the HRS model is as follows: with the Hawkes process, we can model the positive impact of the occurrence of a past event, and link future video request rates with the past video request events; with the reactive process, we can take the influence of negative events (e.g., the removal of a video from the recommendation list) into account; with the self-correcting process, we can restrict the growth of video popularity. Compared to NN (neural network)-based models, the number of parameters of HRS is much less. In addition, our HRS model is completely explainable and interpretable, making HRS superior in two aspects: 1) HRS is composed of multiple components and each component can be evaluated separately. Thus, it is easier to refine HRS to generalize its use in different platforms [21]. 2) The performance obtained with HRS is more convincing and not due to coincidence [22], [23].

In summary, our main contributions in this paper can be summarized as below:

- We develop a hybrid multi-point process model called *HRS* to accurately predict the evolution of video popularity (i.e., video request rate). Different from NN-based model, our HRS model is completely explainable and interpretable. The HRS model can link future video request rates with both past video request events and negative events, and take the characteristics of edge servers into account.
- We propose an online video caching algorithm for edge servers based on the HRS model. Due to much less model parameters, the algorithm has very low computation complexity. All parameters of the HRS model can be determined automatically by maximizing the Log-likelihood function. Thus, the algorithm can be executed frequently to update cached videos timely according to the dynamics of video popularity.
- We conduct extensive real trace-driven experiments to validate the effectiveness of our proposed algorithm. The video request traces are collected from Tencent Video, one of the largest online video providers in China. The experimental results show that the HRS-based online caching algorithm can achieve the highest cache hit rate with 12.3% improvement than the best baseline algorithm in terms of cache hit rate. In particular, the improvement is over 24% when the caching capacity is very limited. In addition, the execution time of our algorithm is much lower than that of NN-based caching algorithms.

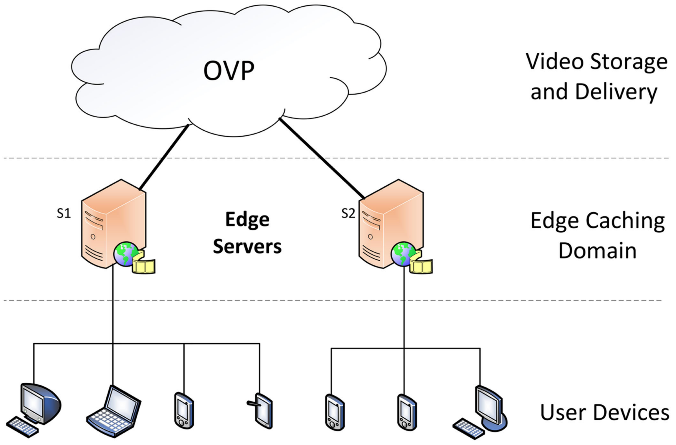


Fig. 1. The system architecture of the video caching system with multiple edge servers.

The rest of the paper is organized as follows. We first provide an introduction of preliminary knowledge in the next section which is followed by the description of the HRS model in Section 3. Notably, the new online caching algorithm is proposed in Section 4. The experimental results are presented in Section 5; while the related works in this area are discussed in Section 6 before we finally conclude our paper.

2 Preliminary

2.1 Video Caching On Edge

We consider a video caching system with multiple edge servers. The system architecture is illustrated in Fig. 1. Online video providers (OVP) store a complete set of videos with population C. A number of edge servers are deployed in the proximity of end users. Since edge servers are closer to end users, serving users with edge servers can not only reduce the Internet traffic but also improve the user QoE [5]. In our scenario, every edge server provides online video services for users in a certain area, e.g., a city, exclusively. To guarantee the computation efficiency, the video caching algorithm on an edge server only utilizes request events recorded by itself. Although exchanging information between edge servers may further improve the caching performance a little bit, it involves additional communication and computation overhead, which is harmful for the efficiency.

TABLE 1
Notations Used in the Paper

Notation	Description
C	The total number of videos stored in OVP.
S	The total number of videos cached on an edge server.
K	The total number of request records.
$\mathcal{E}$	The event set of all requested records.
ε^t / ζ^t	The timestamp set of all requested records / negative events before time t.
ε_i^t / ζ_i^t	The timestamp set of requested records / negative events of video i before time t.
τ / τ'	The timestamp of any request record / negative event.
$\lambda_i(t)$	The conditional intensity function of video i at time t.
$\tilde{\lambda}_i(t)$	The estimation of $\lambda_i(t)$, which is defined by Eq. (7) for HRS.
$\hat{\lambda}_i(t)$	The positive form of $\tilde{\lambda}_i(t)$ adjusted by $g(x) = s\log(1+\exp(x/s))$.
β_i	The bias of intensity function for HRS associated with video i.
ω_i / α_i / γ_i	The parameter of SE / SC / SR term in HRS associated with video i respectively.
$k_0(t-\tau)$ / $k_1(t-\tau')$	The exponential kernel function of SE / SR term reflecting the influence of past event defined by Eq. (8).
δ_0 / δ_1	The decay parameter of $k_0(t-\tau)$ / $k_1(t-\tau')$, which can be determined through cross validation in experiments.
T	The entire observation period time for the computation of likelihood function.
$\boldsymbol{\theta}$	The parameters matrix of a point process and $\theta = [\ \beta^{\intercal}, \omega^{\intercal}, \alpha^{\intercal}, \gamma^{\intercal}\ ]$ in HRS model.
$\theta_i^{(j)}$	The substitute of any parameter associated with video i after j iterations in gradient descent algorithm, such as $\beta_i^{(j)}$.
$ll(\boldsymbol{\theta})$	The Log-likelihood function of point process, which is defined by Eq.(10) for HRS.
$\bar{ll}(\boldsymbol{\theta})$	The evaluated Log-likelihood function of HRS, which is estimated by Monte Carlo method and defined by Eq. (13).
ρ_β / ρ_ω / ρ_α / ρ_γ	The regularization parameter of β_i / ω_i / α_i / γ_i, which can be determined through cross validation in experiments.
M	The number of sample times in Monte Carlo method.
$\mathbf{t}^{(m)}$	The timestamp of the m-th sample in Monte Carlo method.
$\Phi_i(t)$ / $\Psi_i(t)$ / $\Gamma_i(t)$	The kernel function of HRS defined by Eq. (22).
Δt	The time interval of online kernel functions update.
ΔT	The time interval of online parameter update.
ΔM	The number of sample times of online parameter update.
k_{th}	The threshold to truncate the sum of k_0 of online parameter update.

The problem is to predict the video request rates on each edge server in the future so that the most popular videos can be cached in time by the edge server. Therefore, our objective is to maximize the cache hit rate on each edge server, which is defined as the number of requests served by the edge server divided by the total number of requests. This objective can be transformed to maximize the prediction accuracy of future video request rates.

Without loss of generality, we consider the video caching problem for a particular edge server, which can store S videos with $S < C$. To simplify our analysis, we assume that all videos are of the same size. Note that it is possible that OVP splits a video into multiple video segments of equal size to simplify management and reduce the implementation cost such as [24], [25]. A video segment is the unit for caching. Our algorithm is still applicable in this case as long as the request events of each video segment are recorded. For simplicity, we use video and video segment interchangeably hereafter.

With this simplification, it is apparent that the top S most frequently requested videos should be cached on each edge server, and our main task is to predict which S videos will be most popular in the future. For convenience, major notations used in this paper are summarized in Table 1.

We first define the event sets as follows. All past video requests are denoted by an event set $\mathcal{E} = \{e_1, e_2, ..., e_K\}$. Let $\varepsilon = \{\tau_1, \tau_2, ..., \tau_K\}$ denote the occurrence time of all past events, $\tau_1 < \tau_2 <, \ldots, < \tau_K$. In other words, events are recorded according to their occurrence time points. Each event in the set $\mathcal{E}$ is a tuple, *i.e.*, $e = \langle \tau, i \rangle$, where τ is the request time and i is the video index. Besides, we define the set ε_i^t as the timestamp set for historical events of video i before time t and $\varepsilon^t = \cup_{\forall i} \varepsilon_i^t$.

2.2 Point Process

Point process is a family of models which are generated from individual events to capture the temporal dynamics of event sequences via the conditional intensity function[26]. We employ the point process models to predict the future request rate given the historical request events of a particular video. In general, the predicted request rate of video i can be defined as $\lambda_i(t|\varepsilon_i^t)$, where ε_i^t is the timestamp set of historical requested records of video i before time t. The conditional intensity function represents the expected instantaneous request rate of video i at time t. In the rest of the paper, we use $\lambda_i(t)$ to represent $\lambda_i(t|\varepsilon_i^t)$.

We first introduce three typical point process models before we specify the expression of $\lambda_i(t)$,

2.2.1 *Hawkes Process (HP)*

HP [18] is also called self-exciting process. For HP, the occurrence of a past event will positively affect the arrival rate of the same type of events in the future. Given the timestamp set of historical events ε_i^t, the arrival rate of such events at time t can be predicted according to the following formula:

$$\lambda_i(t) = \beta_i + \sum_{\forall \tau \in \varepsilon_i^t} \phi_i(t-\tau). \tag{1}$$

Here, β_i is the deterministic base rate and $\phi_i(t-\tau)$ is the kernel function to reflect the influence of the past event at time τ on the arrival rate of the same type event at time t. Moreover, $\phi_i(t-\tau)$ should be a monotonically decreasing function with $t-\tau$ in that more recent events have higher positive influence on the future request rate.

2.2.2 Reactive Process

The reactive process [19] is an extension of HP by linking the future event with more than one types of events. HP only considers the influence of positive events, while the reactive process models both exciting and restraining effects of historical events on instantaneous intensity. The future rate can be given by:

$$\lambda_i(t) = \beta_i + \sum_{\forall \tau \in \varepsilon_i^t} \phi_i^{exc}(t-\tau) - \sum_{\forall \tau' \in \zeta_i^t} \phi_i^{res}(t-\tau'). \tag{2}$$

Here ε_i^t denotes the same timestamp set of positive events as that in HP while ζ_i^t represents the timestamp set of negative events restraining the intensity function of video i before time t. ϕ_i^{exc} and ϕ_i^{res} are kernel functions to reflect the influence of positive and negative events respectively.

2.2.3 Self-Correcting Process

Compared to the Hawkes process and reactive process, the intensity function of the self-correcting process is more stable over time [20]. Once an event occurs, the intensity function will be reduced by a factor $e^{-\alpha_i}$. Here α_i is a parameter representing the ratio of correcting. Mathematically, the intensity function can be given by

$$\lambda_i(t) = \exp[\mu_i t - \alpha_i N_i(t)], \tag{3}$$

where μ_i is the rate of the steadily increase of intensity function and $N_i(t)$ is the number of historical requests of video i until time t. Generally, the time series of events based on the self-correcting process are more uniform than those based on other processes such as HP.

2.2.4 Log-Likelihood Function of Point Process

To apply the point process models to predict video request rates, we need to determine the parameters (denoted by θ) defined in each process, such as μ and α in the self-correcting process. An effective approach to determine these parameters is to maximize the likelihood of the occurrence of a target event set in the entire observation time $(0, T]$, *i.e.*, $\mathcal{E}^T$.

Given the overall intensity function $\lambda(t) = \sum_{\forall i} \lambda_i(t)$ and the occurrence time τ_l of the last event in the historical event time set ε^t, the probability that no event i occurs in the period $[\tau_l, t)$ is $P\{no\ event\ in\ [\tau_l, t) \mid \varepsilon^t\} = \exp\left[-\int_{\tau_l}^{t} \lambda(x)\mathrm{d}x\right]$. Thus, the probability density that an event i occurs at time t is given by:

$$P\{t, i \mid \varepsilon^t\} = \lambda_i(t)\exp\left[-\int_{\tau_l}^{t} \lambda(x)\mathrm{d}x\right]. \tag{4}$$

The detailed derivation can be found in [27]. Given the event set $\mathcal{E}^T = \{e_1, \ldots, e_K\}$, where $e_k = \langle \tau_k, i_k \rangle$ and K is the total number of events during the time interval $(0, T]$, it is easy to derive the likelihood function for a given event set using.

$$\begin{aligned} L(\theta \mid \mathcal{E}^T) &= \prod_{k=1}^{K} \lambda_{i_k}(\tau_k)\exp\left[-\int_{\tau_{k-1}}^{\tau_k} \lambda(t)\mathrm{d}t\right] \\ &\quad \times \exp\left[-\int_{\tau_k}^{T} \lambda(t)\mathrm{d}t\right], \\ &= \prod_{\forall i} \prod_{\tau \in \varepsilon_i^T} \lambda_i(\tau) \times \exp\left[-\int_0^T \lambda(t)\mathrm{d}t\right]. \end{aligned} \tag{5}$$

For convenience, we let $\tau_0 = 0$ and align all time series for different videos i to the same initial point $\tau_0 = 0$. In practice, it is difficult to manipulate the likelihood function. Equivalently, we can optimize the Log-likelihood function, which is defined as:

$$ll(\theta \mid \varepsilon^T) = \sum_{\forall i} \sum_{\forall \tau \in \varepsilon_i^T} \log(\lambda_i(\tau)) - \int_0^T \lambda(t)\mathrm{d}t, \tag{6}$$

where ε^T is the timestamp set of target events in the entire observation time interval $(0, T]$.

3 HRS Model

In this section, we describe the HRS model which is the combination of three point process models introduced in the last section.

3.1 Intensity Function Design

First of all, we explain the intuition behind the HRS model, and the reasons why we take three types of point processes into account.

- *Self-exciting*: If a video has attracted a number of user requests recently, each request can impose some positive influence on the future request rate of the same video. This is the self-exciting process depicted by the Hawkes process. It can well capture videos that are becoming more and more popular.
- *Reactive process*: Different from the positive influence of historical request events, there also exist events that impose negative influence on the future request rates. For example, the popularity of a video may sharply drop down if the video is removed from the recommendation list. Such negative events can be modeled by the reactive process.
- *Self-correcting*: The user population covered by an edge server is limited. Thus, if users do not repeatedly request videos, a video can not stay popular forever. In fact, users seldom replay videos they have watched [11]. It is expected that the popularity of a video will diminish with time after a majority of users have requested it. The restriction of the limited user population can be captured by the self-correcting process.

$$\begin{aligned} \tilde{\lambda}_i(t) = \underbrace{\beta_i}_{\text{bias}} + \overbrace{\omega_i \sum_{\tau \in \varepsilon_i^t} k_0(t-\tau)}^{\text{self-exciting}} \underbrace{\exp[-\alpha_i N_i(\tau)]}_{\text{self-correcting}} \\ \underbrace{- \gamma_i \sum_{\tau' \in \zeta_i^t} k_1(t-\tau'),}_{\text{self-restraining}} \end{aligned} \tag{7}$$

Based on the above discussion, we can see that the evolution of video popularity is a very complicated process. It is difficult to precisely model the video request rate by merely utilizing only one particular kind of point process. In view of that, we propose to construct the video request intensity function by combing three types of point processes together. The proposed request rate intensity function is presented by Eq. (7), where $\lambda_i(t)$ is the true request rate of video i at time t, and $\tilde{\lambda}_i(t)$ is the estimation of $\lambda_i(t)$. Each term in Eq. (7) is explained as follows.

- The first term β_i is the bias of the intensity function for video content i with positive value ($\beta_i > 0$).
- As we have marked in Eq. (7), $\omega_i \sum_{\tau\in\varepsilon_i^t} k_0(t-\tau)$ is the SE (self-exciting) term. It is designed based on the Hawkes process and $\omega_i k_0(t-\tau)$ is the positive influence imposed by the video request event at time τ. $k_0(t-\tau)$ is a kernel function to be specified later and ω_i is a parameter to be learned.
- The second term $\gamma_i \sum_{\tau'\in\zeta_i^t} k_1(t-\tau')$, which is called SR (self-restraining) term, captures the influence of negative events such as the event that a video is removed from the recommendation list. This SR term is designed based on the reactive process. ζ_i^t is the set including the timestamp of all negative events in the period $[0,t)$. Similar to modeling the influence of positive events, $k_1(t-\tau')$ is the kernel function to account for the influence of a negative event. γ_i is a parameter to be learned.
- The last term $\exp[-\alpha_i N_i(\tau)]$ is the SC (self-correcting) term and $N_i(\tau)$ is the number of historical requests of video i until time τ. The implication is that the influence of a positive event will be smaller if more users have watched the video i. For example, if there are two movies: A and B. Movie A has been watched by 99% users, but Movie B is a new one watched by only 1% users. Then, the influence of a request event for Movie A or B should be very different.

For point process models, it is common to adopt exponential kernel functions to quantify the influence of historical events[18], [26], [27]. Thus, in the HRS model, the kernel functions $k_0(t-\tau)$ and $k_1(t-\tau')$ are set as:

$$\begin{aligned} k_0(t-\tau) &= exp[-\delta_0(t-\tau)], \\ k_1(t-\tau) &= exp[-\delta_1(t-\tau')]. \end{aligned} \tag{8}$$

Here $\delta_0 > 0$ and $\delta_1 > 0$ are two hyper-parameters, which can be determined empirically through cross validation. From Eq. (8), we can observe that kernel functions decay with $t-\tau$, implying that the influence gradually diminishes with time.

By considering the reality of video request rates, we need to impose restrictions on the intensity function defined in Eq. (7).

- The video request rate is non-negative. However, due to the SR term, it is not guaranteed that Eq. (7) always yields a non-negative request rate. Besides, the Log-likelihood function requires that the video request rate must be positive. Thus, we define

$$\hat{\lambda}_i(t) = s\log\left(1+\exp(\tilde{\lambda}_i(t)/s)\right). \tag{9}$$

  Here s is a small positive constant number. We utilize the property that the function $g(x) = s\log(1+\exp(x/s)) \approx \max\{0,x\}$, *i.e.*, the ReLU function, as $s\to 0$ [28].
- All parameters β_i, ω_i, α_i and γ_i should be positive numbers to correctly quantify the influence of each term.

The intensity function $\hat{\lambda}_i(t)$ is the final estimation form of the request intensity λ_i^t for video i at time t.

3.2 Maximizing Log-Likelihood Function

Given the event sets ε^T and ζ^T, and the parameters $\boldsymbol{\theta} = [\,\boldsymbol{\beta}^\intercal, \boldsymbol{\omega}^\intercal, \boldsymbol{\alpha}^\intercal, \boldsymbol{\gamma}^\intercal\,]$, the Log-likelihood function is defined in Eq. (10) according to Eq. (6). In the rest of this work, we use the shorthand notation $ll(\boldsymbol{\theta}) =: ll(\boldsymbol{\theta}\,|\,\varepsilon^T, \zeta^T)$ if event sets are clear in the context. In addition, to simplify our notations, let $\mathcal{I}_i = \int_0^T \hat{\lambda}_i(t)\,\mathrm{d}t$ represent the integral term in the Log-likelihood function of video i.

$$\begin{aligned} ll(\boldsymbol{\theta}\,|\,\varepsilon^T,\zeta^T) &= \sum_{\forall i}\sum_{\tau\in\varepsilon_i^T}\log\hat{\lambda}_i(\tau) - \sum_{\forall i}\int_0^T \hat{\lambda}_i(t)\,\mathrm{d}t \\ &= \sum_{\forall i}\sum_{\forall\tau\in\varepsilon_i^T}\log\hat{\lambda}_i(\tau) - \sum_{\forall i}\mathcal{I}_i. \end{aligned} \tag{10}$$

Remark: It is worth mentioning that HRS is a kind of machine learning based caching algorithm. HRS can be adopted by different online video platforms providing assorted video categories since all parameters in HRS are determined automatically by minimizing the loss function defined in Eq. (10). Even if user request patterns vary with these platforms, their patterns can be well gauged by our HRS model with different parameters.

The challenge to maximize the Log-likelihood function lies in the difficulty to derive $\mathcal{I}_i$ due to the complication of Eq. (9). Thus, we resort to the Monte Carlo estimator[29], [30] to derive $\mathcal{I}_i$ approximately.

We briefly introduce the Monte Carlo estimator as follows. Given a function $f(x)$, M samples of x can be uniformly and randomly selected from the domain of x, say (a,b). Then, the integral of $f(x)$ can be calculated as:

$$\begin{aligned} \mathcal{I} &= \int_a^b f(x)\,\mathrm{d}t = \mathbb{E}\left[\bar{\mathcal{I}}^M\right], \\ \text{s.t. } \bar{\mathcal{I}}^M &= \frac{1}{M}\sum_{m=1}^{M}\frac{f(\mathbf{x}^{(m)})}{p(\mathbf{x}^{(m)})}, \\ \mathbf{x}^{(m)} &\sim \mathrm{U}(a,b), \\ p(\mathbf{x}^{(m)}) &= \frac{1}{b-a}, \end{aligned} \tag{11}$$

where a and b are lower and upper limit points of the integral function respectively. The expected value of the integral term, *i.e.*, $\mathcal{I} = \mathbb{E}\left[\bar{\mathcal{I}}^M\right]$, can be approximately performed by the average of $f(\mathbf{x}^{(m)})/p(\mathbf{x}^{(m)})$.

Suppose the integral range of $\mathcal{I}_i$ is from 0 to T, we can apply the Monte Carlo estimator to evaluate $\mathcal{I}_i$ as follows:

$$\begin{aligned} \mathcal{I}_i &\approx \frac{T}{M}\sum_{n=1}^{M}\hat{\lambda}_i\left(\mathbf{t}^{(m)}\right), \\ \text{s.t. } \mathbf{t}^{(m)} &\sim \mathrm{U}(0,T). \end{aligned} \tag{12}$$

The Log-likelihood function can be approximately evaluated by:

$$\begin{aligned}\bar{ll}(\boldsymbol{\theta}) &= \sum_{\forall i}\sum_{\forall \tau\in\varepsilon_i^T}\log\hat{\lambda}_i(\tau) - \sum_{\forall i}\bar{\mathcal{I}}_i^M,\\ &= \sum_{\forall i}\sum_{\forall \tau\in\varepsilon_i^T}\log\hat{\lambda}_i(\tau) - \frac{T}{M}\sum_{\forall i}\sum_{m=1}^{M}\hat{\lambda}_i\left(\mathbf{t}^{(m)}\right),\\ &\qquad \text{s.t.}\quad \mathbf{t}^{(m)}\sim \mathrm{U}(0,T).\end{aligned}\tag{13}$$

Equivalently, we can minimize the negated Log-likelihood function. By involving the regularization terms, our problem can be formally defined as:

$$\begin{aligned}&\min_{\boldsymbol{\theta}}\ \mathcal{L} = -\bar{ll}(\boldsymbol{\theta}) + \rho_\beta\|\boldsymbol{\beta}\|_2 + \rho_\omega\|\boldsymbol{\omega}\|_2 + \rho_\alpha\|\boldsymbol{\alpha}\|_2 + \rho_\gamma\|\boldsymbol{\gamma}\|_2,\\ &\text{s.t.}\quad \beta_i,\omega_i,\alpha_i,\gamma_i > 0,\ \text{for } \forall i,\end{aligned}\tag{14}$$

where $\rho_\beta, \rho_\omega, \rho_\alpha$ and ρ_γ are regularization parameters.

Since the Log-likelihood in Eq. (13) is a convex function, we can solve Eq. (14) by using the Gradient Descent (GD) algorithm [26], [27]. By differentiating $\bar{ll}(\boldsymbol{\theta})$ with respect to each parameter $\theta_i \in \boldsymbol{\theta}$,[1] one can derive the following results:

$$\frac{\partial\,\bar{ll}(\boldsymbol{\theta})}{\partial\theta_i} = \sum_{\tau\in\varepsilon_i^T}\frac{1}{\hat{\lambda}_i(\tau)}\frac{\partial\hat{\lambda}_i(\tau)}{\partial\theta_i} - \frac{T}{M}\sum_{m=1}^{M}\frac{\partial\hat{\lambda}_i(\mathbf{t}^{(m)})}{\partial\theta_i}.\tag{15}$$

According to Eq. (15), we need to derive $\partial\hat{\lambda}_i(t)\,/\,\partial\theta_i$ in order to obtain the gradient of $\bar{ll}(\boldsymbol{\theta})$. Thus, by differentiating $\hat{\lambda}_i(t)$ shown in Eq. (9) with respect to each parameter, we can obtain Eqs. (16), (17), (18), and (19) as follows:

$$\frac{\partial\hat{\lambda}_i(t)}{\partial\beta_i} = \frac{\partial\hat{\lambda}_i(t)}{\partial\tilde{\lambda}_i(t)},\tag{16}$$

$$\frac{\partial\hat{\lambda}_i(t)}{\partial\alpha_i} = -\frac{\partial\hat{\lambda}_i(t)}{\partial\tilde{\lambda}_i(t)}\omega_i\sum_{\tau\in\varepsilon_i^t}k_0(t-\tau)N_i(\tau)\exp[-\alpha_i N_i(\tau)],\tag{17}$$

$$\frac{\partial\hat{\lambda}_i(t)}{\partial\omega_i} = \frac{\partial\hat{\lambda}_i(t)}{\partial\tilde{\lambda}_i(t)}\sum_{\tau\in\varepsilon_i^t}k_0(t-\tau)\exp[-\alpha_i N_i(\tau)],\tag{18}$$

$$\frac{\partial\hat{\lambda}_i(t)}{\partial\gamma_i} = -\frac{\partial\hat{\lambda}_i(t)}{\partial\tilde{\lambda}_i(t)}\sum_{\tau'\in\zeta_i^t}k_1(t-\tau').\tag{19}$$

Here $\partial\hat{\lambda}_i(t)/\partial\tilde{\lambda}_i(t)$ can be calculated by Eq. (20), which gives

$$\frac{\partial\hat{\lambda}_i(t)}{\partial\tilde{\lambda}_i(t)} = \frac{\exp(\tilde{\lambda}_i(t)/s)}{1+\exp(\tilde{\lambda}_i(t)/s)}.\tag{20}$$

By integrating Eqs. (16), (17), (18), (19), (20) with Eq. (15), we can obtain the gradient expression of $\bar{ll}(\boldsymbol{\theta})$. Then, let $\theta_i^{(j)}$ represent any parameter to be determined after j iterations. We update θ_i according to:

$$\theta_i^{(j+1)} \leftarrow \theta_i^{(j)} + \eta\left(-\frac{\partial\,\bar{ll}(\boldsymbol{\theta})}{\partial\,\theta_i^{(j)}} + \rho_{\theta_i}\theta_i^{(j)}\right),\tag{21}$$

1. θ_i is a parameter associated with video i, such as β_i.

where ρ_{θ_i} is the regularization parameter associated with the parameter θ_i and η is the learning rate depending on the GD algorithm. In our work, to adhere to Eq. (21) and ensure all parameters are within the boundaries specified in Eq. (14), we adopt the L-BFGS [31] algorithm to minimize the objective function.

3.3 Analysis of Computation Complexity

The computation complexity for a round of iteration to determine the parameters in the HRS model is $O(|\varepsilon^T| + |\zeta^T| + CM)$, where $|\varepsilon^T|$ and $|\zeta^T|$ are the total numbers of positive and negative events respectively, C is the total number of videos and M is the Monte Carlo sampling number of each video.

We analyze the detailed computation complexity as follows. To minimize the objective function in Eq. (14), it is necessary to compute gradients according to Eqs. (16), (17), (18), and (19). To ease our discussion, we define three functions as below:

$$\Phi_i(t) = \sum_{\tau\in\varepsilon_i^t}k_0(t-\tau)\exp[-\alpha_i N_i(\tau)]\tag{22a}$$

$$\Psi_i(t) = \sum_{\tau'\in\zeta_i^t}k_1(t-\tau')\tag{22b}$$

$$\Gamma_i(t) = \sum_{\tau\in\varepsilon_i^t}k_0(t-\tau)N_i(\tau)\exp[-\alpha_i N_i(\tau)]\tag{22c}$$

We need to compute ε_i^T for different $\Phi_i(t)$'s and $\Gamma_i(t)$'s, and ζ_i^T for $\Psi_i(t)$'s. Specifically, for a particular event with occurrence time τ and its previous event with occurrence time τ_l, we can compute $\Phi_i(\tau) = \Phi_i(\tau_l)exp[-\delta_0(\tau-\tau_l)] + \exp[-\alpha_i N_i(\tau)]$.[2] Thus, the computation complexity is $O(1)$ to compute $\Phi_i(t)$ for each event and the complexity is $O(|\varepsilon_i^T|)$ to complete the computation for all video i events. Recalling that $\varepsilon^T = \cup_{\forall i}\varepsilon_i^T$, the whole computation complexity for all $\Phi_i(t)$'s is $O(|\varepsilon^T|)$. Similarly, the computation complexity is $O(|\varepsilon^T|)/O(|\zeta^T|)$ to compute all $\Psi_i(t)$'s/$\Gamma_i(t)$'s.

With the above computations, the complexity to compute the first term of Eq. (15) is $O(|\varepsilon^T|)$. For the Monte Carlo estimator, it needs to sample M times for each video. Thus, there are totally CM samples in a round of iteration for C videos. Besides, given the sampled time point $\mathbf{t}^{(m)}$, the complexity is $O(1)$ to compute all gradients. By wrapping up our analysis, the overall computation complexity for each iteration is $O(|\varepsilon^T| + |\zeta^T| + CM)$.[3]

4 Online HRS Based Video Caching Algorithm

The complexity $O(|\varepsilon^T| + |\zeta^T| + CM)$ is not high if the training algorithm is only executed once. Yet, the online video system is dynamic because user interests can change over time rapidly, and fresh videos (or users) enter the online video system continuously. The computation load will be too heavy if we need to update the predicted video request rates very frequently. Thus, in this section, we propose an online video caching algorithm based on the HRS model.

2. Here, we utilize the property that $k_0(0) = 1$.

3. In fact, this complexity is an upper bound since we can complete the computation of $\Psi_i(t)$ in the first iteration without the necessity to update it in subsequent iterations.

The online algorithm can update the predicted video request rates by minor computation with incremental events since the last update.

Algorithm 1. Kernel Function Online Update Algorithm

```
Input: $\Delta t$ , $\{\varepsilon^{t+\Delta t} - \varepsilon^{t}\}$ , $\{\zeta^{t+\Delta t} - \zeta^{t}\}$ , $\Phi_i(t)$'s , $\Psi_i(t)$'s , $N_i(t)$'s
Output: $\Phi_i(t+\Delta t)$'s , $\Psi_i(t+\Delta t)$'s , $N_i(t+\Delta t)$'s
1: while $\forall i$ do
2:     $\Phi_i(t+\Delta t) \leftarrow \Phi_i(t) \cdot exp[-\delta_0 \Delta t]$
3:     $\tau_l \leftarrow t$
4:     for $\forall \tau \in \{\varepsilon_i^{t+\Delta t} - \varepsilon_i^{t}\}$ do
5:         $N_i(\tau) \leftarrow N_i(\tau_l) + 1$
6:         $\Phi_i(t+\Delta t) \leftarrow \Phi_i(t+\Delta t) + k_0(t+\Delta t-\tau) \cdot \exp[\alpha_i N_i(\tau)]$
7:         $\tau_l \leftarrow \tau$
8:     end
9:     $N_i(t+\Delta t) \leftarrow N_i(\tau_l)$
10:    $\Psi_i(t+\Delta t) \leftarrow \Psi_i(t) \cdot exp[-\delta_1 \Delta t]$
11:    for $\forall \tau' \in \{\zeta_i^{t+\Delta t} - \zeta_i^{t}\}$ do
12:        $\Psi_i(t+\Delta t) \leftarrow \Psi_i(t+\Delta t) + k_1(t+\Delta t-\tau')$
13:    end
14: end
15: return $\Phi_i(t+\Delta t)$'s , $\Psi_i(t+\Delta t)$'s , $N_i(t+\Delta t)$'s
```

The online video caching algorithm needs to cope with two kinds of changes. The first one is the kernel function update. Given the HRS model parameters, the video request rates predicted according to Eq. (7) should be updated according to the latest events. The user interest can be very dynamic. For example, users may prefer News videos in the morning, but Movie and TV videos in the evening. Thus, the predicted request rates should be updated instantly and frequently in accordance with the latest events. The second one is the parameter update. The model parameters such as ω_i and γ_i capture the influence weight of each term in the HRS model. In a long term, due to the change of users and videos, model parameters should be updated as well. We discuss the computation complexity to complete the above updates separately.

4.1 Online Update of Kernel Functions

According to Eq. (7), if there are new events, we need to update kernel functions, *i.e.*, $\Phi_i(t)$ and $\Psi_i(t)$, so as to update $\lambda_i(t)$. Suppose that the time point of the last update is t and the current time point to update request rates is $t+\Delta t$. Then, the computation complexity to complete the update is $O(|\varepsilon^{t+\Delta t} - \varepsilon^{t}| + |\zeta^{t+\Delta t} - \zeta^{t}|)$. In other words, the complexity is only a linear function with the number of incremental events.

For using exponential kernel functions, we can prove that

$$k_0(t+\Delta t-\tau) = k_0(t-\tau)exp[-\delta_0 \Delta t], \tag{23}$$
$$k_1(t+\Delta t-\tau) = k_1(t-\tau)exp[-\delta_1 \Delta t]. \tag{24}$$

Note that the term $exp[-\alpha_i N_i(\tau)]$ in Eq. (7) is not dependent on t. Thus, we can complete the update of terms $\omega_i \Phi_i(t)$ and $\gamma_i \Psi_i(t)$ with $O(1)$ by multiplying $exp[-\delta_0 \Delta t]$ and $exp[-\delta_1 \Delta t]$ respectively. Then, we only need to add $|\varepsilon_i^{t+\Delta t} - \varepsilon_i^{t}| + |\zeta_i^{t+\Delta t} - \zeta_i^{t}|$ for each video i. Note that it is unnecessary to update videos without any new event. Thus, the overall computation complexity is $O(|\varepsilon^{t+\Delta t} - \varepsilon^{t}| + |\zeta^{t+\Delta t} - \zeta^{t}|)$. The algorithm details for updating kernel functions are shown in Algorithm 1.

Algorithm 2. Parameter Online Learning Algorithm

```
Input: $\theta^{(0)}$ , $T$ , $\Delta T$, $\{\varepsilon^{T+\Delta T} - \varepsilon^{T+\frac{\ln k_{th}}{\delta_0}}\}$ , $\{\zeta^{T+\Delta T} - \zeta^{T}\}$ , $\Psi_i(T)$
Output: $\theta^{(j)}$
1: Update $\Psi_i(t)$ where $t \in [T, T+\Delta T)$ once at first
2: $j \leftarrow 0$
3: while The termination condition is not satisfied do
4:     Update $\Phi_i(t)$,$\Gamma_i(t)$ where $t \in [T, T+\Delta T)$ with set $\{\varepsilon^{T+\Delta T} - \varepsilon^{T+\frac{\ln k_{th}}{\delta_0}}\}$
5:     $l \leftarrow 0$
6:     $\nabla \leftarrow [0]^{4*C}$
7:     while $\forall i$ do
8:         for $\Delta M$ samples then
9:             $\mathbf{t}^{(m)} \sim \mathrm{Unif}(T, T+\Delta T)$
10:            Calculate $\hat{\lambda}_i(\mathbf{t}^{(m)})$, $\tilde{\lambda}_i(\mathbf{t}^{(m)})$ according to Eq. (7) and
               Eq. (9) with $\Phi_i(\mathbf{t}^{(m)})$ and $\Psi_i(\mathbf{t}^{(m)})$
11:            $l \leftarrow l - \hat{\lambda}_i(\mathbf{t}^{(m)})$
12:            for $\theta_i \in \{\beta_i, \omega_i, \alpha_i, \gamma_i\}$ do
13:                Calculate $\partial\hat{\lambda}_i(\mathbf{t}^{(m)}) \,/\, \partial\theta_i$ by one of equation in
                   Eqs. (16)-(19) corresponding to $\theta_i$
14:                $\nabla_{\theta_i} \leftarrow \nabla_{\theta_i} - \partial\hat{\lambda}_i(\mathbf{t}^{(m)}) \,/\, \partial\theta_i$
15:            end
16:        end
17:        $l \leftarrow (\Delta T/\ \Delta M) \cdot l$
18:        $\nabla \leftarrow (\Delta T/\ \Delta M) \cdot \nabla$
19:        for $\forall \tau \in \{\varepsilon_i^{T+\Delta T} - \varepsilon_i^{T}\}$ do
20:            $l \ \leftarrow l + \log \hat{\lambda}_i(\tau)$
21:            for $\theta_i \in \{\beta_i, \omega_i, \alpha_i, \gamma_i\}$ do
22:                $\nabla_{\theta_i} \leftarrow \nabla_{\theta_i} + (\partial\hat{\lambda}_i(\tau) \,/\, \partial\theta_i)/\ \hat{\lambda}_i(\tau)$
23:            end
24:        end
25:    end
26:    Update $\theta^{(j+1)} \leftarrow \theta^{(j)}$ by adopting L-BFGS algorithm with
       $l$, $\nabla$ and the penalty term
27:    $j \leftarrow j+1$
28: end
29: $T \leftarrow T + \Delta T$;
30: return $\theta^{(j)}$
```

4.2 Online Update of Parameters

To update parameters in accordance with new events, it is necessary to update gradients based on Eqs. (16), (17), (18), and (19) and execute the GD algorithm again to obtain updated parameters. To avoid confusing with the kernel function update, we suppose the time point of the last parameter update is T and the current time point is $T + \Delta T$[4].

In Eqs. (16), (17), (18), and (19), we can also see kernel functions $\Phi_i(t)$'s, $\Psi_i(t)$'s and $\Gamma_i(t)$'s. Therefore, the update of them will be firstly introduced. If maintaining a fixed value during the learning process, the kernel functions $\Psi_i(t)$'s can be simply updated by incremental new events based on the discussion in the last subsection. However, as for $\Phi_i(t)$'s and $\Gamma_i(t)$'s, we need to scan all historical events again to compute their values when the parameter α has been updated, which results in very high computation complexity. To reduce the additional complexity, we propose to use a threshold k_{th} to truncate the sum of kernel functions. k_{th} is a very small number. If $k_0(t-\tau) < k_{th}$ where t is the

4. Note that the update frequency of parameters is different from that of video request rates.

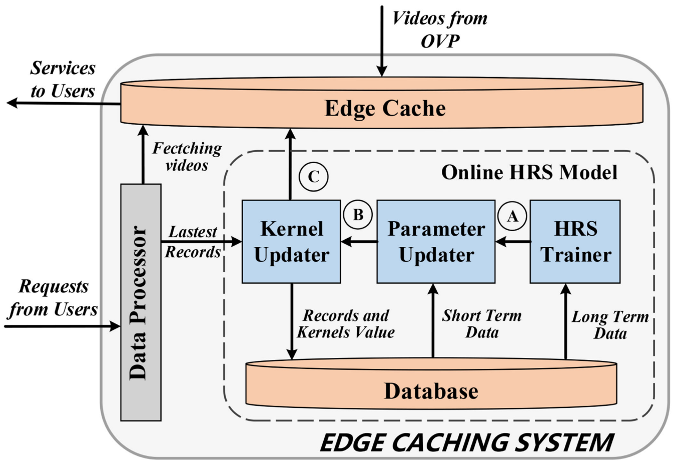


Fig. 2. Modules of an edge cache system implementing online HRS algorithm. A representative workflow is presented in order.

current time and τ is the occurrence of a particular event, it implies that the influence of the historical event before time τ is negligible. Therefore, it is trivial to ignore this event so that the computation complexity will not continuously grow with time. Given k_{th} and the current update time $T + \Delta T$, it is easy to verify that events between time $[T + \frac{\ln k_{th}}{\delta_0}, T + \Delta T)$ needs to be involved to update $\Phi_i(t)$'s and $\Gamma_i(t)$'s. Thus, the upper bound of computation complexity for kernel renewal is $O(|\varepsilon^{T+\Delta T} - \varepsilon^{T+\frac{\ln k_{th}}{\delta_0}}| + |\zeta^{T+\Delta T} - \zeta^{T}|)$.

Next, we introduce the update of all gradients in Eq. (15). The first term relevant to recent new events can be updated in $O(|\varepsilon^{T+\Delta T} - \varepsilon^{T}|)$. Furthermore, the Monte Carlo sampling term needs to be trimmed during online learning process, *i.e.*, $\frac{\Delta T}{\Delta M}\sum_{m=1}^{\Delta M}\frac{\partial\hat{\lambda}_i(\mathbf{t}^{(m)})}{\partial\theta_i}$. We suppose there are ΔM new samples during the period from $[T, T + \Delta T)$, where $\Delta M/M = \Delta T/T$ since we update the parameters in a shorter time window. Therefore, we can conclude that the complexity to calculate the Monte Carlo sampling term is $O(C \cdot \Delta M)$ with updated kernels. Here C is the total number of videos in the system.

By wrapping up our analysis, the overall computation complexity to update all gradients is $O(|\zeta^{T+\Delta T} - \zeta^{T}| + |\varepsilon^{T+\Delta T} - \varepsilon^{T+\frac{\ln k_{th}}{\delta_0}}| + |\varepsilon^{T+\Delta T} - \varepsilon^{T}| + C \cdot \Delta M)$ in each iteration. Note that this is also an upper bound of the computation complexity. In the end, we present the detailed online learning algorithm for training the HRS model in Algorithm 2.

4.3 Framework of HRS Edge Caching System

In Fig. 2, we describe the framework of our system including an *Online HRS model*, a *Data Processor* and an *Edge Cache*.

As shown in Fig. 2, the *Data Processor* is responsible for preprocessing the request records. It is also responsible for recording positive and negative events. The *Online HRS model* utilizes the user request records from the *Data Processor* to periodically update kernel functions and the HRS model parameters. The *Edge Cache* updates the cached videos based on the update of prediction results generated by the HRS model.

To take both computation overhead and bandwidth consumption into account, the *Edge Cache* can periodically rerank video request rates with new prediction updated by the *Online HRS model*. If a newly requested video is not hit by the edge cache, it makes a decision to cache the video based on the updated prediction. As a result, the *Edge Cache* maintains videos with the highest request rates in its cache and evicts videos with the lowest request rates when the cache is fully occupied.

The *Online HRS model* can be further decomposed into three parts, including *HRS Trainer*, *Parameter Updater* and *Kernel Updater*. A typically workflow through these three parts is shown as follows:

A) This is the first step to deploy the HRS algorithm on an edge node. The HRS model can be initially trained by the *HRS Trainer* with the long term request records according to the method introduced in Section 3.2.

B) With the arrival of new video requests, the parameters such as α_i and β_i in the HRS model can be refined by the *Parameter Updater* according to Algorithm 2. Thus, these parameters reflect not only long-term but also short-term popularity trends of videos. To avoid overfitting, these parameters should not be updated over frequently. In our experiments, we update parameters for every a few number of days.

C) The user view interest may change very quickly with time, which can be captured by the renewal of prediction on video request rates through frequently updating the kernel functions. With the incremental user requests, the *Kernel Updater* can efficiently update the kernel functions to trace the latest user request interest.

5 Evaluation

We evaluate the performance of our HRS algorithm by conducting experiments with real traces collected from the Tencent Video.

5.1 Dataset

Tencent video[5] is one of the largest online video streaming platforms in China. We collected a total of 30 days of request records from Nov 01, 2014 to Nov 30, 2014. After data cleaning, encoding and masking, we randomly sample a dataset which contains a population $C = 20K$ of unique videos from 5 cities in Guangdong province in China. There is a total number of $K = 15.84M(15,841,209)$ request records in this dataset and we make the dataset publicly available on GitHub.[6] Each request record in our dataset is represented by the metadata $\langle\, \mathit{VideoID}, \mathit{UserID}, \mathit{TIME}, \mathit{PROVINCE}, \mathit{CITY} \,\rangle$. Given the lack of negative events, the set of negative events is generated from request records as follows: if a video stays cold without being requested for a period, a negative event of this video will be marked in the dataset. Empirically, the period is set as 12 hours.

We divided the dataset into five parts based on the date of request records for cross-validation and hyper-parameters selection following the Forward-Chaining trick[32].

5. Tencent Video: https://v.qq.com
6. https://github.com/zhangxzh9/HRS_Datasets

Each part includes the request records in six days. In the first fold, Part I, including the request traces in the first 6 days, is used as the training set; while Part II, with the records from day 6 to 12, is used as validation set. Part III, including the request records of the next 6 days, is used as the test set. In the second fold, Part I and II together serve as the training set, while Part III and IV are used as the validation set and test set respectively. Finally, we employ Parts I-III as the training set and the rest two parts as the validation and test set respectively in the third fold.

5.2 Evaluation Metrics

We employ three metrics for performance evaluation.

- *Cache Hit Rate* is defined as the number of requests hit by the videos cached on the edge server divided by the total number of requests issued by users. If the HRS runs independently on multiple edge servers, the overall cache hit rate of the whole system will be calculated by the weighted average hit rates of multiple edge servers.
- *Bandwidth Consumption Rate (BCR)* is defined as the fraction of bandwidth consumed by OVP to serve all requests. If a requested video is not cached by the edge server, the request will be served by OVP. Since the edge server is closer to end users, a lower BCR is more desirable. BCR is formally defined as:

$$BCR = \frac{the\ number\ of\ requests\ served\ by\ OVP}{the\ total\ number\ of\ requests}$$

- *Execution Time*: Due to the possibility that the edge server is with very limited computing resource, it is desirable that the computation load of the caching algorithm is under control so that cached videos can be updated timely. Thus, we use the execution time of each algorithm as the third evaluation metric.

5.3 Baselines

We compare the performance of HRS with the following baselines:

- *LRU (Least Recently Used)*, which always replaces the video that was requested least recently with the newly requested one when the cache is full.
- *OPLFU (Optimal Predictive Least Frequently Used)*[13], which is a variant of LFU. Different from LFU, it predicts the future popularity by matching and using one of Linear, Power-Law, Exponential and Gaussian functions. Caching server maintains the cache list based on the estimated future popularity determined by the selected function.
- *POC (Popcaching)* [24], which learns the relationship between the popularity of videos and the context features of requests, and stores all features in the Learning Database for video popularity prediction. Once a request arrives, POC will update the features of the requested video online and predicts video popularity by searching the Learning Database with the context features. we set the number of requests in the past 1 hour, 6 hours, 1 day as the first three features while 10 days, 15 days and 20 days as the fourth dimension feature for three folds, respectively.
- *LHD (Least Hit Density)* [33], which is a rather rigorous eviction policy to determine which video should be cached. LHD predicts potential hits-per-space-consumed (hit density) for all videos using conditional probability and eliminates videos with the least contribution to the cache hit rate. An public implementation of the LHD algorithm in GitHub[7] can be obtained.
- *DPC (DeepCache)* [17], which predicts video popularity trend by leveraging the LSTM Encoder-Decoder model. An M-length input sequence with d-dimensional feature vector representing the popularity of d videos is required for the model. A K-length sequence will be exported for prediction. Here M and K are hyper-parameters for model. All model settings are the same as the work [17] in our experiments.
- *Optimal (Optimal Caching)*, which is achieved by assuming that all future requests are known so that the edge server can always make the optimal caching decisions. It is not a realistic algorithm, but can be used to evaluate the improvement space of each caching algorithm.

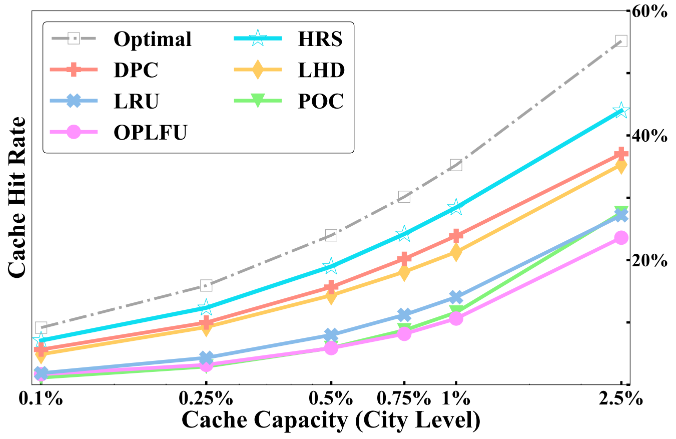
Our principle is to select the most influential or state-of-the-art caching algorithms as baselines. LRU is one of the most classical caching algorithms and its principle has been adopted by real systems. OPLFU is an variant of LFU, which is also a classical algorithm adopted by previous works [2], [24], [34]. POC, DPC and LHD are more advanced popularity-based caching algorithms. They are designed based on probabilistic models or neural network models, which represent state-of-the-art caching techniques.

5.4 Experimental Settings

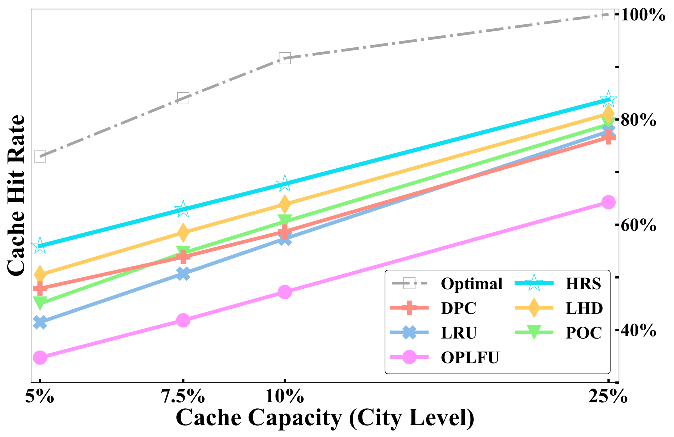
We simulate a video caching system shown in Fig. 2 to evaluate all caching algorithms. In the gradient descent algorithm, there are six hyper-parameters which are initialized as $\delta_0 = 0.5$, $\delta_1 = 1.5$ and $\rho_\beta = \rho_\alpha = \rho_\omega = \rho_\gamma = e^5$ empirically. Their values will be determined through validation. For all parameters $\boldsymbol{\theta}$ in HRS, their initial values are set as 1, except that the initial values of $\boldsymbol{\gamma}$ are set as 0.1, referring to the settings in some previous papers[19], [27], [35]. Moreover, the number of sample times in the Monte Carlo estimation is set as 144,000 times for every day, *e.g.*, we set $M = 1,728,000$ in the first fold, and set $M = 2,592,000$ and $M = 3,456,000$ for the second and third folds respectively. The time interval ΔT to update the online HRS model is set as two days. The iteration process in the gradient descent algorithm will be terminated if the improvement of cache hit rate in the validation set is negligible after an iteration.

By default, the time interval to update kernel functions is set as 1 hour, and the truncating threshold k_{th} is set as e^{-9} for parameter online learning. Some detailed experiments are conducted to study the influence of these two parameters. Furthermore, all algorithms except LHD are programmed with Python[36] and executed in Jupyter

7. https://github.com/CMU-CORGI/LHD



(a) Cache hit rate with small cache at city level.



(b) Cache hit rate with large cache at city level.

Fig. 3. Cache hit rate for HRS and other baseline algorithms when varying the cache capacity (size) from $S = 0.1\%$ (20) to $S = 25\%$ (5000) at the city level.

Notebook [37] with a single process. As for LHD, we reuse the code and estimate the execution time according to [33]. Besides, the execution time is measured on an Intel server with Xeon(R) CPU E5-2670 @ 2.60GHz.

5.5 Experimental Results

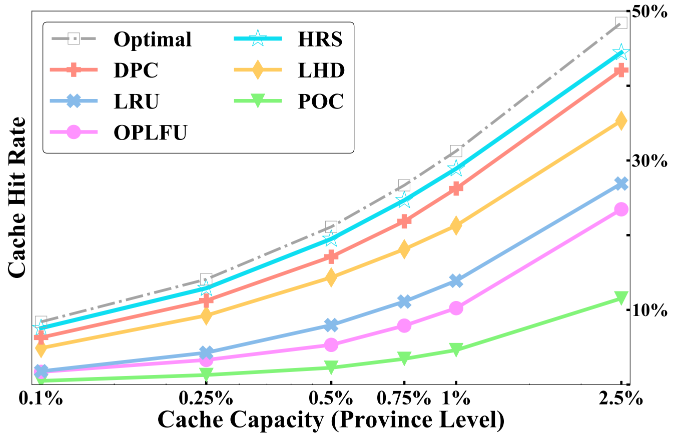
5.5.1 Comparison of Cache Hit Rate

We first evaluate the HRS algorithm with other baseline caching algorithms through experiments by varying the caching capacity from 0.1% to 25% of the total number of videos (*i.e.*, the caching size is varied from 20 to 5K videos). The experiment results of 5 cities are presented in Fig. 3 with the y-axis representing the averaged cache hit rate and Fig. 4 shows the results of cache hit rate of a single server at the province level.

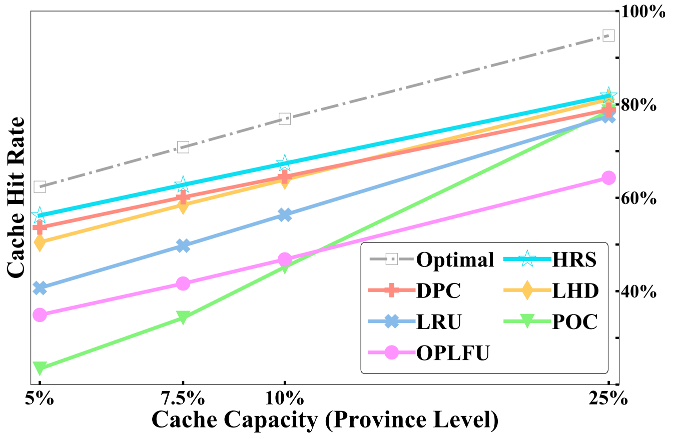
Through the comparison, we can see that:

- HRS outperforms all other baseline algorithms evaluated in terms of the cache hit rate under all cache sizes over the test time, with an overall average of 15.5% improvement at city level. DPC is the second best one in most cases.
- It is more efficient to utilize the caching capacity by using HRS. To demonstrate this point, let us see a specific case with a target cache hit rate of 10% in Fig. 3. In this case, HRS requires a cache size of 40 videos. In comparison, DPC/LHD needs nearly 2-2.4 times cache capacity to achieve the same goal. The performance improvement against the second best solution exceeds 145% with 0.1% limited caching capacity, showing the outstanding ability of HRS to predict the most popular video when the resource is constrained and a more accurate decision is needed.
- Compared to other baseline algorithms at the province level, HRS also achieves an overall average of 8.4% improvement. The HRS model performs better at the city level than at the province level for the reason that video popularity trends are more accurately reflected by leveraging the SC term in city edge servers. In fact, the HRS model performs better if the request rate is higher.

Moreover, to check the stability of each video caching algorithm, we plot the cache hit rate over time with a fixed caching capacity $S = 200$ (equivalent to about 1% of total videos). The results are presented in Fig. 5, showing the



(a) Cache hit rate with small cache at province level.



(b) Cache hit rate with large cache at province level.

Fig. 4. Cache hit rate for HRS and other baseline algorithms when varying the cache capacity (size) from $S = 0.1\%$ (20) to $S = 25\%$ (5000) at the province level.

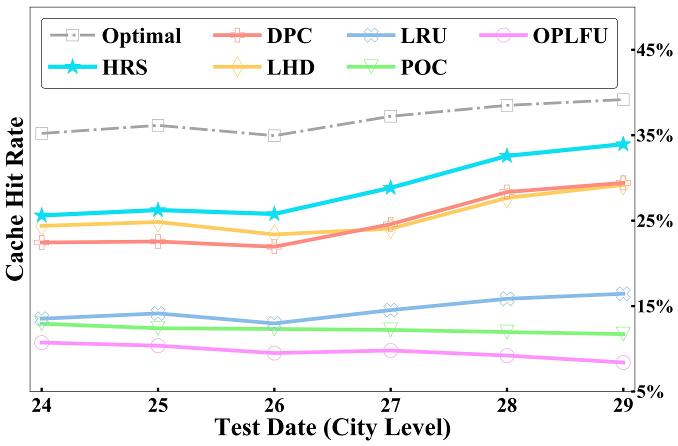


Fig. 5. Average cache hit rate of each day over the test period at city level. Each point in the figure shows the average cache hit rate over a day. The cache capacity is fixed at $S = 1\% \ (200)$.

averaged cache hit rate of all algorithms versus the date. In other words, each point in the figure represents the average cache hit rate over a day. From the results shown in Fig. 5, we can draw a conclusion that HRS is always the best one achieving the highest cache hit rate among video caching algorithms except the Optimal one indicating the gain of HRS is very stable over time.

5.5.2 Comparison of Bandwidth Consumption Rate

To further demonstrate the benefit to OVP with a higher hit rate, we conduct the experiment in Fig. 6 by comparing the BCR (Bandwidth Consumption Rate) of OVP to serve all user requests with different caching algorithms. From the results in Fig. 6, we can observe that the BCR of the HRS algorithm is the best one consuming the least server bandwidth than baselines. Its improvement is significant when the cache capacity is between 0.5% and 7.5%. It is not difficult to understand that all caching algorithms perform very well/bad when the caching capacity is extremely large/small. In particular, by only caching 7.5% video contents, HRS can reduce 60% bandwidth consumption for OVP, indicating that HRS is effective in reducing bandwidth cost for OVP.

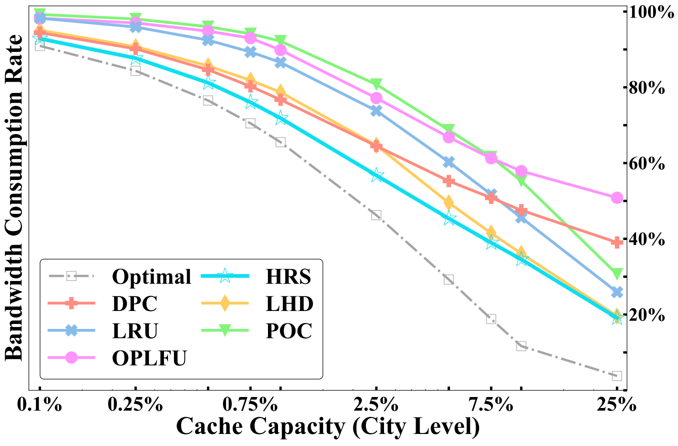


Fig. 6. The reduced rate of bandwidth consumption for HRS and others baseline algorithms when varying the cache capacity (size) from $S = 0.1\% \ (20)$ to $S = 25\% \ (5000)$ at the city level.

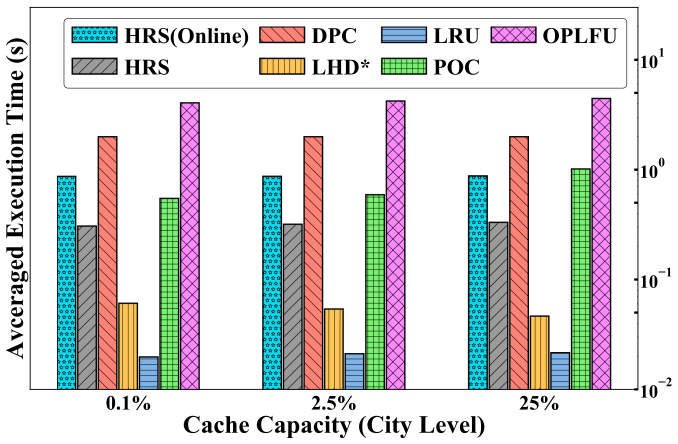


Fig. 7. Comparison of execution time under different cache sizes. Results show the averaged time(s) for each update interval Δt (default $\Delta t = 1$ hour).

5.5.3 Comparison of Execution Time

We conduct experiments to evaluate the averaged time for each video caching algorithm with different caching capacities at the city level. The default time interval to execute each algorithm is one hour, *i.e.*, $\Delta t = 1$h. The experiments are carried out in the all test periods in the three folds to achieve convincing results. The results are presented in Fig. 7. Both HRS (Online) and HRS are conducted to examine the influence of online algorithm on computation complexity.

As we see from Fig. 7, the heuristically designed algorithms, *i.e.*, LHD and LRU, achieve the lowest execution time. However, HRS is the best one compared with other proactive video caching algorithms, *i.e.*, DPC, POC and OPLFU. Compared with the one-hour time interval to execute caching algorithms, the time cost of HRS (Online) algorithms is negligible. For example, it only costs 0.875s per hour with caching size of 5000 (about 25% of total videos). Further, if the computing resource is constrained, HRS (online) can be degraded to HRS, which can achieve 2.7x speed up on average. These experiment results indicate the feasibility of HRS for online edge video caching.

5.5.4 Impact of Each Component

To investigate the contribution to hit rate by each point process component in our HRS model, we conduct ablation experiment by proposing three variants of HRS, which are HS, HR and HPP. HS or HR is a simplified version of the HRS model without considering the SR or SC term, respectively. HPP removes both SR and SC terms with only SE term left. We conduct the experiment at the city level and and the results are shown in Fig. 8.

From the results, we can observe that HRS (online) and HRS outperform their variants. The gap between HRS (Online) and HPP is only moderate indicating that the SE term is the most important component for HRS. HR and HS are a little bit better than HPP indicating that SR and SC terms can slightly improve the caching hit rate.

5.5.5 Setting of Hyper-Parameters

Lastly, We study the sensitivity of two crucial parameters: Δt and k_{th} in Tables 2 and 3 to see how these two hyper-

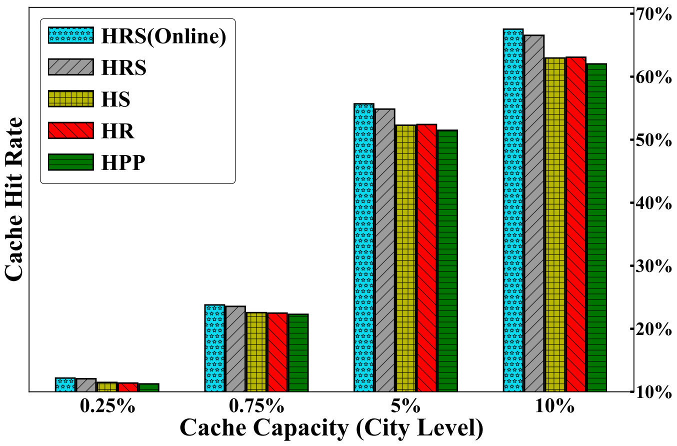


Fig. 8. Cache hit rate of HRS and its variants under different cache sizes for demonstrating the impact of each point process in HRS. The setting is the same as that in Fig. 3.

parameters affect the video caching performance. All other hyper-parameters are kept unchanged as we vary Δt and k_{th}.

We repeat the experiments presented in Fig. 3 by setting different values for Δt. The parameter Δt indicates how frequently the HRS model updates the kernel functions. As we can see in Table 2, the cache hit rate is higher if Δt is smaller because the latest user trends can be captured in time with a smaller Δt. It also confirms that the user interest is very dynamic over time. However, it is more reasonable to set $\Delta t = 1$ hour since the improvement using $\Delta t = 0.5$ hour is small but with higher time complexity .

In Table 3, we further investigate the influence of k_{th} on the cache hit rate. k_{th} is negligible and can be discarded to control the computation complexity. We reuse the setting of the experiment in Fig. 3 except varying k_{th}. As we can see from Table 3, the overall cache hit rate is better if k_{th} is smaller since more kernel functions are reserved for computation. Because the cache hit rate is very close by setting k_{th} equal to e^{-7} or e^{-8}, we finally set $k_{th} = e^{-7}$ in our experiments with lower time complexity.

6 Related Work

6.1 Video Caching

Caching at the edge is an effective way to alleviate the backhaul burden and reduce the response time. In recent years, more and more research interest has been devoted to investigating the caching problem on edge servers. Ma *et al.* conducted a complete analysis of the performance of edge mobile video caching with different factors and proposed a novel geo-collaborative caching strategy [2]. Poularakis *et al.* formulated a joint routing and caching problem guaranteed

TABLE 2
Cache Hit Rate(%) Under Different Δt (hour(s))

Update	Cache Capacity						
Interval	0.1%	0.25%	0.5%	1%	2.5%	5%	25%
$\Delta t = 3.0$	6.157	10.964	17.270	26.622	42.338	54.644	82.747
$\Delta t = 1.0$	7.096	12.353	**18.975**	28.450	**43.969**	**55.960**	**83.771**
$\Delta t = 0.5$	**7.223**	**12.386**	18.874	**29.070**	42.754	54.595	83.312

The setting is the same as that in Fig. 3 except Δt.

TABLE 3
Cache Hit Rate(%) Under Different Values for k_{th}

Truncating	Cache Capacity						
Threshold	0.1%	0.25%	0.5%	1%	2.5%	5%	25%
$k_{th} = e^{-6}$	7.096	12.351	18.974	28.432	43.961	55.955	83.756
$k_{th} = e^{-7}$	7.096	12.353	**18.975**	**28.450**	43.969	**55.960**	**83.771**
$k_{th} = e^{-8}$	**7.105**	**12.357**	**18.975**	**28.450**	**43.970**	55.923	83.732

The setting is the same as that in Fig. 3 expect k_{th}.

by an approximation algorithm to improve the percentage of requests served by small base stations (SBSs)[3]. Jiang *et al.* developed a cooperative cache and delivery strategy in heterogeneous 5G mobile communication networks [4]. Yang *et al.* devised an online algorithm which estimates future popularity by location customized caching schemes in mobile edge servers [5]. Moreover, with the ability to learn the context-specific content popularity online, a context-aware proactive caching algorithm in wireless networks was introduced by Muller *et al.* [6].

With the explosive growth of video population, it is urgent to develop more intelligent video caching algorithms by identifying popularity patterns in historical records. It was summarized in [38] and [39] that diverse approaches for content caching have been implemented in the Internet nowadays. However, less attention has been allocated to optimize the caching methods and most of them were deployed based on heuristic algorithms such as LRU, LFU and their variants [12], [13], [14], which are lightweight but inaccurate, and thus often fail to seize viewers' diverse and highly dynamic interests.

Some proactive models including regression models [40], auto regressive integrated moving average [41] and classification models [42] were proposed to forecast the popularity of content. Moreover, quite a few learning-driven caching algorithms were proposed for some special application scenarios. Wu *et al.* proposed an optimization-based approach with the aim to balance the cache hit rate and cache replacement cost[43]. Wang *et al.* developed a transfer learning algorithm to model the prominence of video content from social streams [44], while Roy *et al.* proposed a novel context-aware popularity prediction policy based on federated learning [45].

Besides, with the rapid development of deep learning, a significant amount of research efforts has been devoted to predicting content popularity using the neural network model. Tanzil *et al.* adopted a neural network model to estimate the popularity of contents and select the physical cache size as well as the place for storing contents[46]. LSTM was also implemented for content caching in [16], [17]. Furthermore, deep reinforcement learning (DRL) with the aim to maximize the caching efficiency through interactions with the environment was advocated in [47], [48], [49]. However, NN-based models typically require a large number of historical records for tuning the extensive parameters. But with the sparse requested records of cold videos, it is not easy to learn an appropriate model for prediction.

Further, because the popularity distribution of contents may constantly change over time [7], [8], [11], it is difficult to make decisions based on outdated dataset. Thus, some online learning models which are more responsive to continuously changing trends of content popularity were proposed in [5], [6], [24].

6.2 Point Process

The point processes are frequently used to model a series of superficially random events in order to reveal the underlying trends or predict future events [50]. Bharath *et al.*[51] considered a learning-driven approach with independent Poisson point processes in a heterogenous caching architecture. Shang *et al.* [52] formulated a model to obtain a large-scale user-item interactions by utilizing point process models. Xu *et al.* [35] modeled user-item interactions via superposed Hawkes processes, a kind of classic point process model, to improve recommendation performance. More applications of point processes in recommendation systems can be found in [53], [54]. Furthermore, point processes have been applied to study social networks between individual users and their neighbors in [55]. Ertekin *et al.* [19] used reactive point process to predict power-grid failures, and provide a benefit-and-cost analysis for different proactive maintenance schemes. Mei *et al.* [28] proposed a novel model which was a combination of point processes and neural networks to improve prediction accuracy for future events. The reason why point processes have been applied in predicting discrete events in the future lies in that the occurrence of a past event often gives a temporary boost to the occurrence probability of events of the same type in the future. Naturally, the video request records can be regarded as time series events, which can be modeled by point processes. However, there is very limited work that explored the utilization of point process models to improve video caching decisions, which is the motivation of our work.

7 Conclusion

In this work, we propose a novel HRS model to make video caching decisions for edge servers in online video systems. HRS is developed by combing the Hawkes process, reactive process and self-correcting process to model the future request rate of a video based on the historical request events. The HRS model parameters can be determined through maximizing the Log-likelihood of past events, and detailed iterative algorithms are provided. In view of the dynamics of user requests, an online HRS-based algorithm is further proposed, which can process the request events in an incremental manner. In the end, we conduct extensive experiments through real video traces collected from Tencent Video to evaluate the performance of HRS. In comparison with other baselines, HRS not only achieves the highest cache hit rate, but also maintains low computation overhead.

References

[1] G. Forecast *et al.*, “Cisco visual networking index: Global mobile data traffic forecast update, 2017–2022,” White Paper, 2019.

[2] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, “Understanding performance of Edge content caching for mobile video streaming,” *IEEE J. Sel. Areas Commun.*, vol. 35, no. 5, pp. 1076–1089, May 2017.

[3] K. Poularakis, G. Iosifidis, and L. Tassiulas, “Approximation algorithms for mobile data caching in small cell networks,” *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3665–3677, Oct. 2014.

[4] W. Jiang, G. Feng, and S. Qin, “Optimal cooperative content caching and delivery policy for heterogeneous cellular networks,” *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1382–1393, May 2017.

[5] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, “Content popularity prediction towards location-aware mobile edge caching,” *IEEE Trans. Multimedia*, vol. 21, pp. 915–929, 2019.

[6] S. Muller, O. Atan, M. Van Der Schaar, and A. Klein, “Context-aware proactive content caching with service differentiation in wireless networks,” *IEEE Trans. Wireless Commun.*, vol. 16, no. 2, pp. 1024–1036, Feb. 2017.

[7] S. Dhar and U. Varshney, “Challenges and business models for mobile location-based services and advertising,” *Commun. ACM*, vol. 54, no. 5, pp. 121–129, May 2011.

[8] G. Szabo and B. A. Huberman, “Predicting the popularity of online content,” *Commun. ACM*, vol. 53, no. 8, pp. 80–88, Aug. 2010.

[9] Y. Zhou, J. Wu, T. H. Chan, S. W. Ho, D. M. Chiu, and D. Wu, “Interpreting video recommendation mechanisms by mining view count traces,” *IEEE Trans. Multimedia*, vol. 20, pp. 2153–2165, 2018.

[10] Y. Zhou, L. Chen, C. Yang, and D. M. Chiu, “Video popularity dynamics and its implication for replication,” *IEEE Trans. Multimedia*, vol. 17, pp. 1273–1285, 2015.

[11] J. Wu, Y. Zhou, D. M. Chiu, and Z. Zhu, “Modeling dynamics of online video popularity,” *IEEE Trans. Multimedia*, vol. 18, pp. 1882–1895, 2016.

[12] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, “High performance cache replacement using re-reference interval prediction (RRIP),” *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 60–71, Jun. 2010.

[13] J. Famaey, F. Iterbeke, T. Wauters, and F. De Turck , “Towards a predictive cache replacement strategy for multimedia content,” *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 219–227, Jan. 2013.

[14] M. Z. Shafiq, A. X. Liu, and A. R. Khakpour, “Revisiting caching in content delivery networks,” in *Proc. ACM SIGMETRICS*, 2014, pp. 567–568.

[15] M. Ahmed, S. Traverso, P. Giaccone, E. Leonardi, and S. Niccolini, “Analyzing the performance of LRU caches under non-stationary traffic patterns,” Jan. 2013, *arXiv:1301.4909*.

[16] H. Feng, Y. Jiang, D. Niyato, F. C. Zheng, and X. You, “Content popularity prediction via deep learning in cache-enabled fog radio access networks,” in *Proc. 38th IEEE Global Commun. Conf.*, 2019, pp. 1–6.

[17] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z. L. Zhang, “DEEPCACHE: A deep learning based framework for content caching,” in *Proc. ACM SIGCOMM Workshop NetAI*, 2018, pp. 48–53.

[18] A. G. Hawkes, “Spectra of some self-exciting and mutually exciting point processes,” *Biometrika*, vol. 58, no. 1, Apr. 1971, Art. no. 83.

[19] Ertekin, C. Rudin, and T. H. McCormick, “Reactive point processes: A new approach to predicting power failures in underground electrical systems,” *Ann. Appl. Statist.*, vol. 9, no. 1, pp. 122–144, 2015.

[20] V. Isham and M. Westcott, “A self-correcting point process,” *Stochastic Processes Their Appl.*, vol. 8, no. 3, pp. 335–347, 1979.

[21] A. Dethise, M. Canini, and S. Kandula, “Cracking open the black box: What observations can tell us about reinforcement learning agents,” in *Proc. ACM SIGCOMM Workshop NetAI*, 2019, pp. 29–36.

[22] F. Doshi-Velez and B. Kim, “Towards A rigorous science of interpretable machine learning,” Feb. 2017, *arXiv:1702.08608*.

[23] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?,” Explaining the predictions of any classifier,” in *Proc. 22th ACM Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1135–1144.

[24] S. Li, J. Xu, M. Van Der Schaar, and W. Li, “Popularity-driven content caching,” in *Proc. 35th IEEE Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[25] H. Mao, R. Netravali, and M. Alizadeh, “Neural adaptive video streaming with pensieve,” in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2017, pp. 197–210.

[26] D. Daley, J. Daryl, and Vere-Jones, *An Introduction to the Theory of Point Processes: Volume II: General Theory and Structure*. New York, NY, USA: Springer, 2008.

[27] M.-A. Rizoiu, Y. Lee, S. Mishra, and L. Xie, “A Tutorial on Hawkes processes for events in social media,” Aug. 2017, *arXiv:1708.06401*.

[28] H. Mei and J. Eisner, “The neural Hawkes process: A neurally self-modulating multivariate point process,” in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6755–6765.

[29] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning," *Mach. Learn.*, vol. 50, no. 1–2, pp. 5–43, Jan. 2003.

[30] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," *J. Mach. Learn. Res.*, vol. 21, pp. 1–62, 2020.

[31] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, Sep. 1995.

[32] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Inf. Sci.*, vol. 191, pp. 192–213, May 2012.

[33] N. Beckmann, H. Chen, and A. Cidon, "LHD: Improving cache hit rate by maximizing hit density," in *Proc. 15th USENIX Conf. Netw. Syst. Des. Implementation*, 2018, pp. 389–403.

[34] Y. Zhou, L. Chen, C. Yang, and D. M. Chiu, "Video popularity dynamics and its implication for replication," *IEEE Trans. Multimedia*, vol. 17 pp. 1273–1285, 2015.

[35] H. Xu, D. Luo, X. Chen, and L. Carin, "Benefits from superposed Hawkes processes," in *Proc. 21st Int. Conf. Artif. Intell. Statist.*, 2018, pp. 623–631.

[36] "Pypy," 2001. [Online]. Available: https://www.python.org/

[37] "Jupyter," 2014. [Online]. Available: https://jupyter.org/

[38] H. S. Goian, O. Y. Al-Jarrah, S. Muhaidat, Y. Al-Hammadi, P. Yoo, and M. Dianati, "Popularity-based video caching techniques for cache-enabled networks: A survey," *IEEE Access*, vol. 7, pp. 27 699–27 719, 2019.

[39] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain, "Overlay networks: An Akamai perspective," *Adv. Content Del. Streaming Cloud Serv.*, vol. 51, no. 4, pp. 305–328, Oct. 2014.

[40] Z. Wang, L. Sun, C. Wu, and S. Yang, "Enhancing internet-scale video service deployment using microblog-based prediction," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 775–785, Mar. 2015.

[41] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand forecast and performance prediction in peer-assisted on-demand streaming systems," in *Proc. 30th IEEE INFOCOM*, 2011, pp. 421–425.

[42] M. Rowe, "Forecasting audience increase on YouTube," in *Proc. Workshop User Profile Data Social Semantic Web 8th Extended Semantic Web Conf.*, 2011, pp. 1–15.

[43] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. Lau, "Scaling social media applications into geo-distributed clouds," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 689–702, Jun. 2015.

[44] S. D. Roy, T. Mei, W. Zeng, and S. Li, "Towards cross-domain learning for social video popularity prediction," *IEEE Trans. Multimedia*, vol. 15 pp. 1255–1267, 2013.

[45] Y. Wu, Y. Jiang, M. Bennis, F. Zheng, X. Gao, and X. You, "Content popularity prediction in fog radio access networks: A federated learning based approach," in *Proc. 39th IEEE INFOCOM*, 2020, pp. 1–6.

[46] S. M. 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, 2017.

[47] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, Nov. 2018.

[48] J. Luo, F. R. Yu, Q. Chen, and L. Tang, "Adaptive video streaming with edge caching and video transcoding over software-defined mobile networks: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 1577–1592, Mar. 2020.

[49] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network Edge: A multi-agent deep reinforcement learning approach," in *Proc. 39th IEEE Conf. Comput. Commun.*, 2020, pp. 2499–2508.

[50] R. Lemonnier, K. Scaman, and A. Kalogeratos, "Multivariate Hawkes processes for large-scale inference," in *Proc. 31st AAAI*, 2017, pp. 2168–2174.

[51] B. N. Bharath, K. G. Nagananda, and H. V. Poor, "A learning-based approach to caching in heterogenous small cell networks," *IEEE Trans. Commun.*, vol. 64, no. 4, pp. 1674–1686, Apr. 2016.

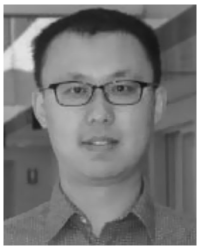
[52] J. Shang and M. Sun, "Local low-rank Hawkes processes for temporal user-item interactions," in *Proc. 18th IEEE Int. Conf. Data Mining*, 2018, pp. 427–436.

[53] N. Du, Y. Wang, N. He, and L. Song, "Time-sensitive recommendation from recurrent user activities," in *Proc. 28th In. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3492–3500.

[54] H. Dai, Y. Wang, R. Trivedi, and L. Song, "Recurrent coevolutionary latent feature processes for continuous-time recommendation," in *Proc. 1st Workshop Deep Learn. Recommender Syst.*, 2016, pp. 29–34.

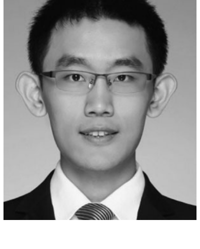
[55] K. Zhou, H. Zha, and L. Song, "Learning triggering kernels for multi-dimensional Hawkes processes," in *Proc. 30th Int. Conf. Int. Conf. Mach. Learn.*, 2013, pp. 1301–1309.

Xianzhi Zhang received the BS degree from Nanchang University (NCU), Nanchang, China, in 2019. He is currently working toward the MS degree with Sun Yat-sen University (SYSU), Guangzhou, China. His current research interests include content caching, applied machine learning, edge computing, and multimedia communication.

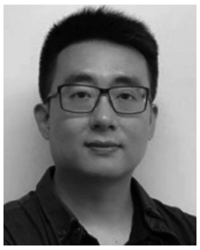


Yipeng Zhou (Member, IEEE) received the bachelor's degree in computer science from the University of Science and Technology of China (USTC), and the PhD and Mphil degrees in information engineering from the Department of CUHK. He is currently a senior lecturer in computer science with the School of Computing, Macquarie University, and the recipient of ARC DECRA in 2018. From Aug. 2016 to Feb. 2018, he was a research fellow with Institute for Telecommunications Research (ITR), University of South Australia. From 2013.9-2016.9, He was a lecturer with the College of Computer Science and Software Engineering, Shenzhen University. He was a postdoctoral fellow with the Institute of Network Coding (INC), The Chinese University of Hong Kong (CUHK) from Aug. 2012 to Aug. 2013. His research interests include federated learning, privacy protection and caching algorithm design in networks. He has published more than 80 papers including IEEE INFOCOM, ICNP, IWQoS, IEEE ToN, JSAC, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Mobile Computing*, and *IEEE Transactions on Multimedia*, etc.

Di Wu (Senior Member, IEEE) received the BS degree from the University of Science and Technology of China, Hefei, China, in 2000, the MS degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2003, and the PhD degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 2007. He was a postdoctoral researcher with the Department of Computer Science and Engineering, Polytechnic Institute of New York University, Brooklyn, NY, USA, from 2007 to 2009, advised by Prof. K. W. Ross. He is currently a professor and the associate dean of the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. He was the recipient of IEEE INFOCOM 2009 Best Paper Award, IEEE Jack Neubauer Memorial Award, and etc. His research interests include edge/cloud computing, multimedia communication, Internet measurement, and network security.



Miao Hu (Member, IEEE) received the BS and PhD degrees in communication engineering from Beijing Jiaotong University, Beijing, China, in 2011 and 2017, respectively. He is currently an associate professor with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. From September 2014 to September 2015, he was a visiting scholar with the Pennsylvania State University, PA. His research interests include edge/cloud computing, multimedia communication, and software defined networks.



Xi Zheng (Member, IEEE) received the PhD degree in software engineering from the University of Texas at Austin, in 2015. From 2005 to 2012, he was the chief Solution Architect for Menulog Australia. He is currently the director of Intelligent Systems Research Group (ITSEG. ORG), senior lecturer (aka associate professor US) and deputy program leader in software engineering, Macquarie University, Sydney, Australia. His research interests include Internet of Things, intelligent software engineering, machine learning security, human-in-the-loop AI, and edge intelligence. He is highly-recommended for the Excellence in Early Career Research Award in Macquarie University (2020) and has a number of highly cited papers. He is awarded multiple highly-competitive Australian Research Council projects since 2019 (ARC LP 19, ARC DP 20, and ARC LP 21). He also serves as PC members for FSE (CORE A*) and PerCom (CORE A*), and PC chair for CPSCom 2021.

Min Chen (Fellow, IEEE) is currently a full professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST) since February 2012. He is the director of Embedded and Pervasive Computing (EPIC) Lab, and the director of Data Engineering Institute at HUST. He is the founding chair of IEEE Computer Society Special Technical Communities on Big Data. He was an assistant professor with the School of Computer Science and Engineering, Seoul National University. He worked as a postdoctoral fellow with the Department of Electrical and Computer Engineering, University of British Columbia for three years. He has more than 300 publications, including more than 200 SCI papers, more than 100 IEEE Transactions/Journals papers, 34 ESI highly cited papers and 12 ESI hot papers. He has published 12 books, including Big Data Analytics for Cloud/IoT and Cognitive Computing (2017) with Wiley. His Google Scholar Citations reached more than 32,500 with an h-index of 86. His top paper was cited more than 3,650 times. He was selected as Highly Cited Researcher since 2018. He got IEEE Communications Society Fred W. Ellersick Prize in 2017, and the IEEE Jack Neubauer Memorial Award in 2019. His research interests include cognitive computing, 5G Networks, wearable computing, big data analytics, robotics, machine learning, deep learning, emotion detection, and mobile edge computing, etc.

Song Guo (Fellow, IEEE) is currently a full professor and associate head (Research & Development) with the Department of Computing, Hong Kong Polytechnic University. He also holds a Changjiang chair professorship awarded by the Ministry of Education of China. He is an Highly Cited researcher (Clarivate Web of Science), and an ACM distinguished member. His research interests include the areas of big data, edge AI, mobile computing, and distributed systems. He coauthored four books, co-edited seven books, and published more than 500 papers in major journals and conferences. He is the recipient of the 2019 IEEE TCBD Best Conference Paper Award, 2018 IEEE TCGCC Best Magazine Paper Award, 2019 and 2017 IEEE Systems Journal Annual Best Paper Award, and other eight best paper awards from IEEE/ACM conferences. His work was also recognized by the 2016 Annual Best of Computing: Notable Books and Articles in Computing in ACM Computing Reviews. His research has been sponsored by RGC, NSFC, MOST, JSPS, industry, etc. He is the editor-in-chief of IEEE Open Journal of the Computer Society and the chair of IEEE Communications Society (ComSoc) Space and Satellite Communications Technical Committee. He was an IEEE ComSoc distinguished lecturer and a member of IEEE ComSoc Board of Governors. He has also served for IEEE Computer Society on Fellow Evaluation Committee, Transactions Operations Committee, editor-in-chief Search Committee, etc. He has been named on editorial board of a number of prestigious international journals like *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Emerging Topics in Computing*, etc. He has also served as chairs of organizing and technical committees of many international conferences.