Received December 18, 2019, accepted January 3, 2020, date of publication January 9, 2020, date of current version January 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2965249

An Incremental Learning Based Edge Caching System: From Modeling to Evaluation

GUANGPING XU[1], (Member, IEEE), BO TANG[1], LIMING YUAN[2], YANBING XUE[2], ZAN GAO[2], SALWA MOSTAFA[3], AND CHI WAN SUNG[3], (Senior Member, IEEE)

[1]School of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300384, China
[2]Tianjin Key Laboratory of Intelligence Computing and Novel Software Technology, Tianjin University of Technology, Tianjin 300384, China
[3]Department of Electrical Engineering, City University of Hong Kong, Hong Kong

Corresponding author: Liming Yuan (yuanleeming@163.com)

This work was supported in part by the NSFC projects under Grant 61971309, Grant 6120123, and Grant 61170301, in part by the NSF Project of Tianjin under Grant 17JCYBJC15600, Grant 18JCYBJC84800, and Grant 18JCYBJC85500, in part by the Scientific Research Program of Tianjin Municipal Education Commission under Grant 2017KJ255, in part by the City University of Hong Kong Project under Grant 7005225, and in part by the Tianjin Research Innovation Project for Postgraduate Students under Grant 2019YJSS059.

ABSTRACT Caches are widely applied to improve data delivery performance in distributed systems like edge networks and content delivery networks (CDNs). We consider caching mechanism in those networks that deliver contents to end users. The challenge comes from the dynamic content distribution problem. The distribution of data popularity is highly skewed and changing over time. Besides, the access pattern of the user requests also varies over time. Some learning algorithms for edge caching problems need to rebuild a new model periodically to adapt to system dynamics, where the knowledge learned from the past is discarded. Besides, each model updating needs a large amount of data, leading to outdated models for consecutive user requests. Inspired by the success of incremental learning approaches in processing massive data in real time, we propose an incremental learning based framework at an edge caching server. The incremental learning algorithm is used to preserve valuable knowledge and to adapt to dynamic workloads faster. We implement our incremental learning based cache system prototype and evaluate its performance under various real-world workloads. The experimental results show that our algorithm can boost cache hit ratio for dynamic workloads compared with the state-of-the-art caching algorithms.

INDEX TERMS Edge caching, incremental learning, system dynamics.

I. INTRODUCTION

Nowadays content delivery networks (CDNs) and edge networks are becoming indispensable architectures of modern communication networks. Research shows that up to 72% of internet traffic will be carried by CDNs by 2022 [1]. In those networks, numerous edge content servers such as CDN leaf nodes and small base stations in 5G networks are distributed geographically. These servers cache contents from backend/original content servers for their end-users in proximity. Then, upon arrival of each user request, an appropriate edge cache server provides cached contents. Caching popular content in edge servers is promising to achieve massive savings in terms of energy and bandwidth [2]. With the help of edge caching, the workloads of back-end content servers can be relieved by reducing duplicated downloads and the user experience can be improved by reducing access latency [3].

One of the key points to enhance the performance of edge caching servers is to maximize the cache hit ratio, *i.e.*, the number of cache hits is divided by the total number of requests on each edge server. The dominant challenges are as follows. In many real-world workloads, the distribution of data popularity is highly skewed. As indicated in [4], approximately 50% to 90% of data objects are infrequently accessed. Moreover, the access patterns of requests may change over time [5]. To challenge the access heterogeneity, traditional caching algorithms often explore the recency and frequency of requests to place data into cache. Typically, least-recently-used (LRU), least-frequently-used (LFU) and their variants are widely used to cache popular objects in CDNs [6].

To better improve the performance of edge cache systems, machine learning based cache management algorithms are extensively studied, since edge servers are equipped with

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

both caching and computing capacities for intelligent caching decisions [7]. Applying machine learning approaches to edge caching is aimed to dynamically cache most popular objects restricted to storage capacity. By mining the access patterns of user requests, the features (e.g., the trends of request frequency) of popular requested objects can be learned. Thus, storing potentially popular objects in edge servers predicted by machine learning models could improve cache efficiency. These algorithms are proposed to boost the cache hit ratio recently.

However, we observe that some of these learning approaches [8], [9], especially those offline learning models, may have the following problems for edge caching. *Firstly*, these methods periodically discard the old model since the model may no longer be suitable for current environment. The knowledge learned from the past data will be discarded completely, even though some of the knowledge might be useful. *Secondly*, the model can not be updated until it has accumulated enough data to rebuild a new model. These learning approaches usually need a large training set and many iterations of training to efficiently train a model. Edge caching servers always face fast changing conditions including unexpected (or even adversarial) access patterns [9], such as the changes of users' preference, sudden heavy workloads by emerging hot news and mobility of users. Moreover, users may be remapped to other edge servers by network controller due to load balance or network congestion [10], which increases the variety of the user access patterns.

Based on these observations, our **motivation** is to build an incremental learning based edge caching framework, which not only keeps recent learned knowledge but also learns new knowledge from a limited amount of training data. Our goal is to explore a learning model by the incremental learning framework to adapt to the changes of the access patterns. As a result, we propose an efficient learning based edge caching algorithm to improve the performance of edge caching.

Inspired by the success of incremental learning approaches in processing non-stationary streaming data, we propose an incremental learning based framework at an edge caching server to relieve the problems mentioned above. The key idea for incremental learning is that continuously add new information into the already constructed model to adapt to the change of the environment. The benefit is to exploit current knowledge and minimize the training time for the model adaptation to new data distribution [11]. With the proposed caching framework, we utilize an incremental learning algorithm (Learn++.NSE [12]) to adapt to dynamic workloads and evaluate its performance.

In summary, our main **contributions** are as follows.

- We present an incremental learning based framework at an edge caching server for the caching problem with the dynamic user access pattern. The incremental learning algorithm is used to preserve valuable knowledge and to adapt to dynamic workloads faster.
- We implement our incremental learning based cache system prototype and evaluate it with several real-world workloads. The experimental results show that our algorithm can boost cache hit ratio for dynamic workloads compared with the state-of-the-art caching algorithms.

The rest of this paper is organized as follows. In section II, some related work of machine learning based edge cache management is presented. In section III, some preliminaries of the incremental learning approach used in our system are introduced. In section IV, the system model is described and our objective is formulated. The implementations of each module of the cache system are discussed in section V. In section VI, the experimental results with real-world traces is shown. Finally, conclusions are drawn in section VII.

II. RELATED WORK

With the widely deployment of CDNs and 5G networks, caching data at edge servers (*e.g.*, small base stations in 5G networks) has attracted more investigate interests recently. The goal is that caching a small number of popular objects at edge caching servers can significantly reduce the workload for duplicated downloads from back-end data servers [13]. Nowadays, research work focuses on the intelligent caching management by applying learning algorithms to improve cache efficiency.

In [14], Li *et al.* propose a popularity-driven content caching replacement method by learning the popularity of content and using it to determine which content it should store and which it should evict from the cache. In the method, popularity is learned in an online fashion and requires no training phase. Further, the work of [15] proposes a Markov cache model that seamlessly adapts to the changes of request patterns in a CDN server. The study in [16] models objects behavior using a conditional probability to predict each object expected hit density and adapt caching behavior in real time. The work in [17] formulates a joint content placement and load balancing under dynamic user request and proposes an online primal-dual algorithm to reduce the system cost.

Moreover, reinforcement learning methods are exploited to improve the content caching. The study in [18] jointly considers global and local popularity demands for a local small base station and proposes a Q-learning based reinforcement learning scheme to learn and adapt to the underlying dynamics of user requests. In [19], Chen *et al.* propose a Deep Q-Network based reinforcement learning based framework at an edge node to improve both short-term and long-term cache hit ratios. The study in [20] proposes a 3D augmented convolutional network to extract time series information and solve the problem of imbalanced data. The study in [21] considers the interaction between a parent caching node and leaf nodes in CDNs and applies deep reinforcement learning to adapt to dynamic evolution of user requests.

At the same time, some studies utilize some complicated learning methods to manage edge caching content by using offline learning models. Narayanan *et al.* [8] use an offline-trained LSTM Encoder-Decoder model to forecast the popularities of objects and to prefetch popular objects. The method needs a large training set to train a deep learning model. If the access pattern changes, the deep learning model will be retrained with much heavy costs. In [9], Berger uses lightweight boosted decision trees to build up a caching decision model, called GDBT. To cope with the change of access patterns, the algorithm rebuilds a new model periodically. However, different from online learning and reinforcement learning, such offline learning strategies may not be able to adapt the model as soon as the change of access patterns. Different from those studies, our work explores a specific incremental learning algorithm called Learn++.NSE to adapt to dynamic workloads for edge caching. The method can preserve prior valuable knowledge and adapt to dynamic workloads faster, which brings a higher caching efficiency.

III. PRELIMINARIES

In this section, we briefly describe preliminaries of the incremental learning and its main challenge of concept drift.

A. INCREMENTAL LEARNING

Different from learning methods used in [8], [9], the incremental learning algorithm continuously predicts input instances and incrementally updates the predicting model after receiving new instance. The model is originally proposed by a small amount of training data, and make predictions for the coming instance. As a note, here an input *instance* corresponds to an object request. The model is continuously updated based on the previous one as more training data arrive. To speed up the model updating process, the training for data can be done by one-pass computation without iterations.

The scheme of the incremental learning can be formalized as follows. For a given sequence of data instances, after extracting its feature X, a decision model is a function H that maps the feature X to the output target variable: $\hat{y} = H(X)$. Therefore an incremental learning algorithm specifies how to build a map function, which calls a prediction model, from a sequence of data instances.

We adopt the terms in [22] to describe the basic procedure of incremental learning to make predictions and update models:

1) *Predict*: When a new instance with feature X^t arrives, a prediction $\hat{y^t}$ is made using the prediction model H^{t-1}.
2) *Diagnose*: After some time called verification latency, the true label y^t is available and the loss can be estimated as $L(\hat{y^t}, y^t)$.
3) *Update*: The instance with feature X^t and true label y^t are used for model updating based on previous model and loss to obtain a new model H^t. In some settings, the losses estimation and the model updating need a batch of instances rather than in one-by-one way.

The incremental learning methods can update the model based on previous one upon the arrival of user requests. As a comparison, offline machine learning approaches periodically discard old models and reconstruct new models by many iterations of training to improve prediction accuracy.

B. CONCEPT DRIFT

The concept drift refers to changes in the conditional distribution of the output (i.e., predictions of the model) given the input features [22]. If the concept drift happens, the old pattern mined from the past data may not be suitable for the new data, leading to wrong decisions or predictions. Concept drift can be formally defined as $\exists X : P_{t_0}(X, y) \neq P_{t_1}(X, y)$, where P_{t_0} denotes the joint distribution between input feature X and target variable y at time t_0 [22]. $P(X, y)$ is composed of two parts as $P(X, y) = P(X) * P(y|X)$. That is, concept drift can be triggered by the changes of both $P(X)$ and $P(y|X)$. The phenomenon of concept drift has been recognized as the root cause of decreased effectiveness in many data-driven information systems such as data-driven early warning systems and data-driven decision support systems [23].

Learning for edge caching from non-stationary data stream is a problem of learning under concept drift. Edge caching nodes face quickly changing conditions that include unexpected (or even adversarial) traffic patterns [9], [24]. Besides, users may be remapped to other edge servers by network controller due to load balance or network congestion [10]. That is, the distribution of trend features of request pattern, $P(X)$, varies with time. Same feature may lead to different popularities for the concept drift. Thus, the distribution of popularity of requested objects $P(y|X)$ given trend features also varies with time. For edge caching scenarios, the faster the prediction model adapts to new distribution, the higher the cache efficiency can be derived.

Learning under concept drift has been a pop research area. So far, many incremental learning techniques have been proposed to solve this problem. Learn++.NSE (nonstationary environments) [12] is an ensemble-based incremental learning algorithm with the passive approach to accommodate the uncertainty of concept drift. Passive approaches do not detect the concept drift in the environment, but rather simply perform an adaptation to model parameters when new data arrive. Ensemble based approaches provide a natural fit to the problem of learning in a non-stationary environment and have many advantages [25]. First, the algorithm can easily incorporate new data into the model when new data are presented, simply by adding new base predictors to the ensemble. Second, it can forget irrelevant knowledge by removing the corresponding base predictors. Third, it can perform well for the reoccurrence of earlier request distribution. To cope with the concept drift, we use Learn++.NSE in our proposed caching framework to adapt to dynamic distribution of content popularity. It regularly updates the model based on previous one after a batch of requests.

IV. SYSTEM MODEL AND PROBLEM DEFINITION

A. EDGE CACHING PROBLEM FORMULATION

Consider a content delivery network (CDN) consists of a core server connected to a set of edge servers via dedicated links.

We assume that the core server has access to $\mathcal{C} = (1, 2, \ldots, C)$ objects each of the same size. Each edge server is equipped with a caching capacity that can cache up to S objects where $S < C$. We assume that each edge server serves M clients requesting objects from the set $\mathcal{C}$.

We assume overall requests are distributed into T time slots sequentially. There are m requests in time slot t denoted as $Req^t = (req_1, req_2, \ldots, req_m)^t$, $1 \leq t \leq T$. Each request is represented by $req_i^t = (p_i, q_i)^t$, $1 \leq i \leq m$, where p_i^t is the timestamp of the request and q_i^t is the ID of the requested object.

In our incremental learning caching framework, a sequence of models $\sigma = (H^1, H^2, \ldots, H^t, \ldots, H^{T-1})$, $1 \leq t \leq T-1$, is generated to adapt to dynamic workloads, where H^t makes cache decisions for requests in Req^{t+1}. At first, Req^1 is used to generate initial model H^1.

The long-term hit ratio for time slots $2 \leq t \leq T$ is defined by Equation (1).

$$R = \frac{\sum_{t=2}^{T} \sum_{i=1}^{m} \mathbb{1}(req_i^t)}{M} \tag{1}$$

where the indicator function $\mathbb{1}(req_i^t)$ is defined as Equation (2).

$$\mathbb{1}(req_i^t) = \begin{cases} 1, & \text{if } q_i^t \text{ has been cached} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Therefore, our objective is to generate a sequence of models σ to maximize the long-term cache hit ratio as defined in Equation (3).

$$\underset{\sigma}{\text{maximize}}\ E[R|\sigma] \tag{3}$$

B. INCREMENTAL LEARNING FOR EDGE CACHING

As we know that the probability distribution of the data access frequency is highly skewed, only a small proportion of data will be accessed with a high probability. Therefore, it could yield a high hit ratio by keeping the most commonly accessed objects in the cache [5]. Based on these observations, the incremental learning algorithm can learn from the access pattern of the most accessed objects and cache potentially popular objects to improve cache hit ratio. We describe the incremental learning model and its auto adaptation based on Learn++.NSE as follows.

Let $D^t = (d_1, d_2, \ldots, d_m)^t$, where $d_i^t = (x_i, y_i)^t$, $1 \leq i \leq m$, x_i^t represents the feature vector for req_i^t and y_i^t represents the popularity label of the requested object. A feature vector x_i^t will be extracted when request req_i^t comes and the model H^{t-1} uses x_i^t to get a cache decision according to Equation (4).

$$H^{t-1}(x_i^t) = \begin{cases} 1, & \text{cache } q_i^t \\ 0, & \text{do not cache } q_i^t \end{cases} \tag{4}$$

H^{t-1} is an ensemble composed of base predictors with different voting weights. The cache decision is obtained as the weighted majority voting of the ensemble members.

In this work, the learning algorithm tries to learn the access patterns of objects within the top-K popularity every time slot t. Popularity label y_i^t can be revealed by ranking the frequency of distinct objects requested in Req^t. The popularity label is defined as follows:

$$y_i^t = \begin{cases} 1, & \text{if } q_i^t \text{ with in top } K \text{ popularity} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

After revealing y_i^t, D^t will be used to update the decision model according to Equation (6).

$$H^t = \pi(H^{t-1}, D^t) \tag{6}$$

where π represents the incremental learning algorithm to update H^t with previous model H^{t-1} and training data D^t.

According to the basic procedure of incremental learning mentioned in section III, the loss of the batch of instances between predictions of model H^{t-1} and true popularity label can be calculated at first. Then D^t is used for adapting learning model based on the previous model and the loss. As a result, H^t will be generated as the model for Req^{t+1} in time slot $t+1$ to adapt to potential changes of the access pattern. According to Equation (3), the incremental learning algorithm generates a sequence of models σs to fit the change of access patterns to improve cache efficiency.

V. SYSTEM FRAMEWORK AND IMPLEMENTATION

In this section, we introduce our proposed incremental learning based edge caching framework. First, we give a brief explanation to the function of each component and the whole workflow. Then, we give a detailed description of the incremental learning mechanism for edge caching.

A. FRAMEWORK OVERVIEW

Our incremental learning based edge caching framework consists of the following four components: *a feature table, an incremental learning (IL) predictor, data cache and backend storage*, which are shown in Fig. 1. The first three components are located at the edge server while the last component is located at the core server. The feature table takes the user requests as the input stream and stores a set of run-time context information for each distinct requested object. The IL predictor uses the information in the feature table as an input to predict whether the requested object to cache or not, and then returns the decision to the cache. While the cache is used to store the objects and if a requested object is not cached it will be retrieved from the backend storage. According to the incremental learning procedure mentioned in previous section, the basic workflow at each time slot t takes place as follows.

For each user request req_i^t in Req^t, the requested object's timestamp is used to update the context information in the feature table and the object is looked up in the cache and returned to the client if the requested object has been cached. Otherwise, the object will be requested from the backend

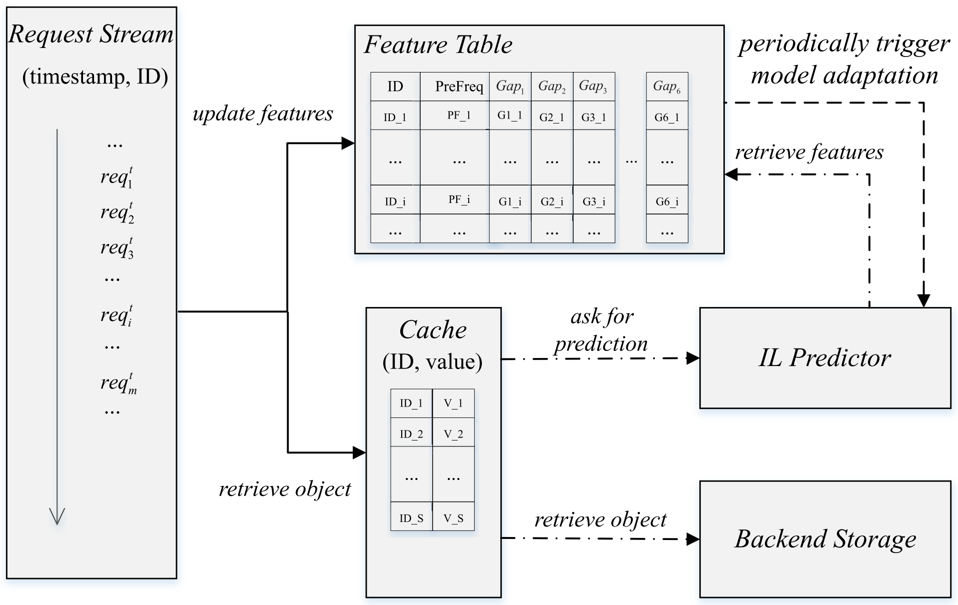

FIGURE 1. **The block diagram of the incremental learning based edge caching framework.**

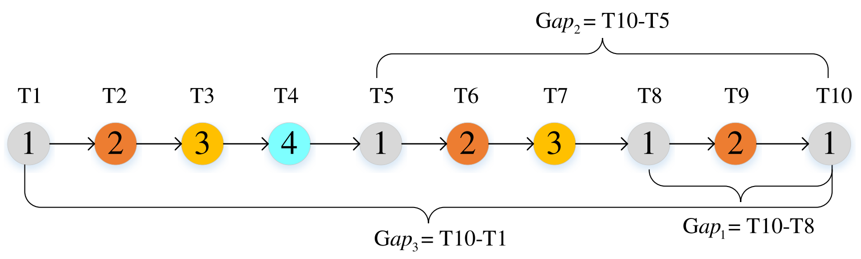

FIGURE 2. **A simple example to illustrate the gaps of accessed objects.**

storage to serve the client. At the same time, the IL predictor is called to get a decision whether to store this object in the cache or not. At the end of each time slot, the decision model will be updated based on the information in the feature table according to incremental learning algorithm.

B. FEATURE TABLE

For each requested object, its context information is mapped into the feature table as input feature vector for the incremental learning algorithm. When a request comes, timestamp gaps between consecutive requests to an identical object are updated. For example, assume that the requested object ID sequence is 1, 2, 3, 4, 1, 2, 3, 1, 2, shown in Fig. 2. When another request for object 1 arrives, the current timestamp gaps for object 1 is denoted as ($Gap_1 = 2, Gap_2 = 5, Gap_3 = 9$). Each object needs a circular queue to maintain its recent requests' timestamps restricted with the queue length. The queue length equals to the length of the feature vector. When a new request to an object arrives, each prior timestamp in the object's circular queue is subtracted by current timestamp to get the feature vector. Then, the timestamp is added into the object's queue. When the queue is full, the oldest buffer will be overlaid by the latest timestamp. We use ($Gap_1, \cdots, Gap_6$) for an identical object as default features. With limited-length queue, features can be refreshed identically and dynamically for each object.

The study in [4] merely uses Gap_1 as the parameter to measure the frequency of a distinct object; here we use up to 6 timestamped gaps to capture the increasing or decreasing trends of popularity. The study in [9] uses machine learning method with consecutive timestamp gaps from 1-50 as features, and each model is trained with $1M$ records. As for our incremental learning approach, we use only $100K$ records for each time slot. Thus, less dimensions of features are used to simplify the model to avoid insufficient training. Based on the idea that if the items kept in the cache are the most commonly accessed ones, the cache has a higher probability resulting into a higher hit ratio [5]. A frequency counter is also needed for identical object to record popularity during the current time slot. When the current time slot ends, the objects are ranked by frequency counters, and then the counters are cleaned. The objects with in top-K frequency are labeled as 1 while others are labeled as 0 according to Equation (5). After the popularity ranking process, features and their corresponding labels are sent to the the IL predictor to trigger model adaptation.

C. IL PREDICTOR

The IL predictor plays the important role in the whole framework, which completes the following two tasks: *prediction* and *adaptation*. For a cache miss, it predicts an object whether within the top-K popularity or not according to Equation (5), and returns the prediction to the cache controller. For a cache hit, the IL predictor just counts it and the hit count will be used to compute the misclassification ratio during model adaptation phase.

When time slot t ends, the IL predictor H^{t-1} will be updated by the model auto adaptation algorithm. All requests in time slot t forms a training set to evaluate and update the model H^{t-1}. The main procedure can be described as follows.

1) For each cache miss in current request stream Req^t, for example req_i^t, the IL predictor will look up the features x_i^t of requested object from the feature table, and then input the features to current prediction model H^{t-1} to make a binary decision to cache it or not.
2) After processing a predefined batch size m of object requests, the model adaptation algorithm is triggered. For adaptation, the algorithm evaluates the loss in current time slot and gets a new ensemble model H^t.
3) The new model is used to predict for requests in time slot $t+1$.

The model auto adaption algorithm is the core of the incremental learning framework. To implement it, there are the following three aspects to consider. *Firstly*, the algorithm gets misclassification ratio to inspect the degree of environment change in current time slot. All instances are weighted by the misclassification ratio to measure their efficiency for training. *Secondly*, the algorithm trains a new base classifier; and then, all existing base classifiers are evaluated on the weighted instances in time slot t. *Finally*, different voting weights are distributed to base classifiers according to their historical performances. The base classifiers with higher performance are more fitted to the recent access pattern. Then, a new model can be derived. A more detailed description of the algorithm is shown in Algorithm 1.

The training set D^t is available at the end of time slot t. Each of m instances of D^t consists of the feature vector x_i^t and the popularity label y_i^t. For $t = 1$, the knowledge base is initialized on the first available batch of data, and the weight w_i^1 for each instance is set to $1/m$. For $t > 1$, the misclassification ratio E^t of the ensemble H^{t-1} is calculated by Equation (7), which reflects the degree of environment change.

$$E^t = 1/m \cdot \sum_{i=1}^{m} \mathbb{1}(H^{t-1}(x_i^t) \neq y_i^t) \tag{7}$$

Then, instance error weights w_i^t are updated according to Equation (8) and are normalized to F_i^t to evaluate the competence of base classifiers. Misclassified instances give a higher weight for better learning to base classifiers than those instances correctly classified. If the misclassification ratio E^t is lower, base classifiers will get higher punishment with misclassified instances for their incompetence. Otherwise,

Algorithm 1 The Model Auto Adaptation Algorithm

```
Input:
 1: Training set $D^t = (x_i, y_i)^t$, $i = 1, \dots, m$ in time slot $t$;
 2: Base classifiers $h_k$, $k = 1, 2, \dots, t-1$ generated in
    previous time slots;
 3: Sigmoid parameter $a$ (slope) and $b$ (inflection point).
Output: New model $H^{t+1}$.
 4: 1.Weight instances:
 5: if $t = 1$ then
 6:     Initialize $F_i^1 = w_i^1 = 1/m, \forall i$
 7: else $F_i^t = w_i^t / \sum_{i=1}^{m} w_i^t, \forall i$
 8: end if
 9: 2.Evaluate base classifiers:
10: Generate a base classifier $h_t$ with $D^t$
11: $\varepsilon_k^t = \sum_{i=1}^{m} F_i^t \cdot \mathbb{1}(h_k(x_i^t) \neq y_i^t)$, for $k = 1, 2, \dots, t$
12: if $\varepsilon_{k=t}^t > 1/2$ then
13:     generate a new base classifier $h_t$
14: end if
15: if $\varepsilon_{k<t}^t > 1/2$ then
16:     set $\varepsilon_k^t = 1/2$
17: end if
18: $\beta_k^t = \varepsilon_k^t / (1 - \varepsilon_k^t)$, for $k = 1, \dots, t$
19: 3.Weight base classifiers and update model:
20: $\bar{\beta}_k^t = \sum_{j=0}^{t-k} \sigma_k^{t-j} \beta_k^{t-j}$, for $k = 1, 2, \dots, t$
21: $W_k^t = log(1/\bar{\beta}_k^t)$, for $k = 1, 2, \dots, t$
22: $H^t(x_i^{t+1}) = \arg max_c \sum_k W_k^t \cdot \mathbb{1}(h_k(x_i^{t+1}) = c)$, for $c \in$
    $\{0, 1\}$
```

it does not need to punish base classifiers.

$$w_i^t = 1/m \cdot \begin{cases} E^t, & H^{t-1}(x_i^t) = y_i^t \\ 1, & \text{otherwise} \end{cases} \tag{8}$$

Next, the training set D^t is used to train a new base classifier, h_t, to adapt to new environment. Note that only base classifiers with error ratio less than $1/2$ can boost ensemble's performance. Thus, the new generated classifier will be retrained if the misclassification ratio is greater than $1/2$. The base classifiers h_k ($k = 1, \cdots, t-1$) are evaluated with weighted instances to get the misclassification ratio ε_k^t in time slot k, and the maximum error is saturated at $1/2$. ε_k^t is normalized between [0, 1] interval as β_k^t, where 0 represents no error, and 1 represents the max error ratio.

For weighting base classifiers, the normalized error β_k^t of each base classifier is weighted by a Sigmoid function which evaluates recent history performance. In the original Sigmoid function as shown in Equation 9, the slope parameter a represents the declining degree of the Sigmoid function and the inflection point parameter b indicates the halfway crossing point. These parameters allow to weight classifiers over a certain scope of times. The Sigmoid function is normalized according to Equation (10).

$$\sigma_k^t = 1/(1 + e^{-a(t-k-b)}) \tag{9}$$

$$\sigma_k^t = \sigma_k^t / \sum_{j=0}^{t-k} \sigma_k^{t-j} \tag{10}$$

Then, the final voting weight W_k^t of each base classifier is derived as a log-normalized form of the weighted error, which is commonly used to update weights of base classifiers in ensemble learning. As a result, the recent competence is considered with more voting weights. Under this sigmoidal weighting strategy, any classifier containing relevant knowledge can receive a high voting weight. Note that it is not the classifier age that affects the voting weight, but the competence determinates its voting weight according to recent environments. Finally, a new model H^t is derived and will be applied in next time slot, where $c = 0$ means not caching the requested object while $c = 1$ means caching according to Equation (2).

To sum up, the model auto adaptation algorithm uses three ways to adapt to changes of the access pattern of requests. *First*, the algorithm gives instances different weights to adapt to the changes and to boost the model performance. *Second*, the algorithm gives base classifiers different voting weights according to their error weights. When access pattern changes, base classifiers with bad performance will be frozen by low voting weights. When the base classifiers fit the access pattern again, they can be activated with high voting weights. *Third*, the voting weights of base classifiers are weighted by the recent performances. Thus, the ensemble can catch up with the latest trends. In this work, the model auto adaptation algorithm is used to update the model in an incremental learning way as the access pattern of requests changes.

D. CACHE CONTROLLER

This component contains cached objects physically. If the requested object has been cached, then it will be served by the cache. The cache management policy is based on the decisions of the IL predictor. If a cache miss happens, the cache controller will fetch the requested object from the backend storage. Meanwhile, it lets the IL predictor to predict whether the object will be in top-K popularity or not. The IL predictor first looks up features x_i^t from feature table and then make a prediction. If the prediction is $H^{t-1}(x_i^t) = 1$, the object will be prefetched into the cache. Here we use LRU as the cache replacement policy.

VI. EXPERIMENTAL EVALUATION

A. SIMULATION SETUP

We implemented the IL predictor based on Learn++.NSE algorithm [26] and the whole prototype was written in JAVA in a CentOS operating system. The measurements are taken on a single core of an Intel i7-7700K CPU running at 4.2GHz with 16GB RAM and one 1TB 7200rpm hard disk. We evaluated the proposed caching algorithm with public real-world workloads.

There are many factors that affect the performance of our algorithm, such as properties of datasets, cache capacities, features for the learning algorithm. In our implementation, the length of each time slot is set to $100K$, and the top-20% popular objects in each time slot are labeled as popular, *i.e.*, $y_i^t = 1$. The access feature takes $(Gap_1, \cdots, Gap_6)$ by default. The base classifier for Learn++.NSE is the naive Bayes, and the Sigmoid parameters are $a = 0.5, b = 10$ as recommend in [12]. Since the halfway crossing point of the Sigmoid function $b = 10$, latest 20 base classifiers are maintained and the old ones are pruned to reduce memory cost.

We compared our proposed algorithm with LRU, LRU-K [27], LFUDA [28] algorithms which are commonly used algorithms in CDNs [28] and GDBT proposed by [9]. LRU-K caches objects after the K hits in an LRU list to filter burst data. In this study, the parameter K is set to 2. LFUDA dynamically ages the frequency counters of LFU to reduce cache pollution problem of LFU. For each trace, another 1 million records are used to warm up the IL model by 10 base classifiers without caching objects, which is enough ensure the learning process to converge. Similarly, 1 million records are also used to warm up a GDBT model. GDBT uses the same settings to train as IL models (the same access features and the top-20% popularity). Each GDBT model is updated every 1 million requests with 30 iterations as done in [9].

TABLE 1. The characteristics of the used datasets to evaluate the performance of caching algorithms.

	# of requests (M)	# of identical objects (M)
CDN	10	2.26
Wikipedia	9	1.59

B. WORKLOAD ANALYSIS

We use two public real-world workloads described in Table 1 to evaluate the performance. Each piece of requests is composed of timestamp and object ID. Another 1M pieces of records to warm up both GDBT and incremental learning models are not included in the table.

- CDN [9] refers to production trace from anonymous top-ten US websites recorded on a SanFrancisco CDN server. We select $10M$ pieces of requests with about $2.3M$ distinct objects.
- Wikipedia [29] is reference to web request to Wikipedia servers. We select 9M pieces of requests with about $1.5M$ identical objects at the beginning of November 2007.

C. PERFORMANCE ANALYSIS

We first show the performance of the proposed algorithm for cache hit ratio under different cache capacities. We vary the cache capacity from 0.8% to 5% of distinct objects. Then, we evaluate the cache hit ratio of our proposed IL algorithm and compare it with other algorithms.

The performance for cache hit ratio with different caching strategies is illustrated in Fig. 3 and Fig. 4. We observe that IL algorithm is always superior to LRU-K, LFUDA and LRU. For CDN trace, IL algorithm performs better than GDBT

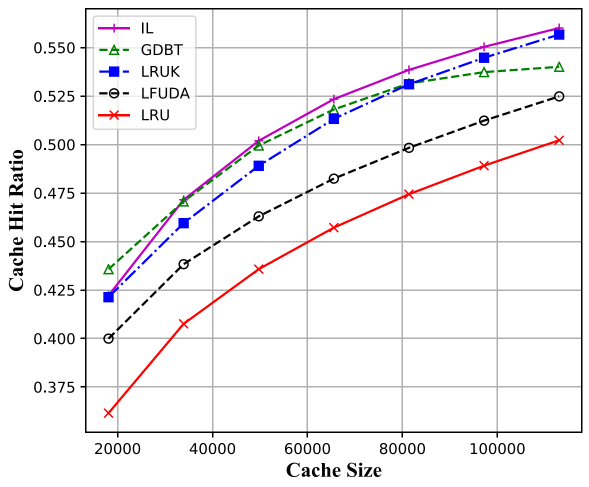

FIGURE 3. Hit-ratio comparison under different caching algorithms driven by CDN trace (IL with the default 6 gaps as feature).

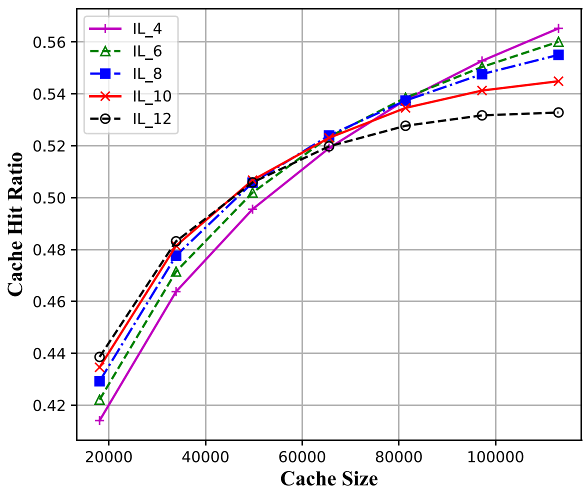

FIGURE 5. Hit-ratios of IL algorithm with different features driven by CDN trace.

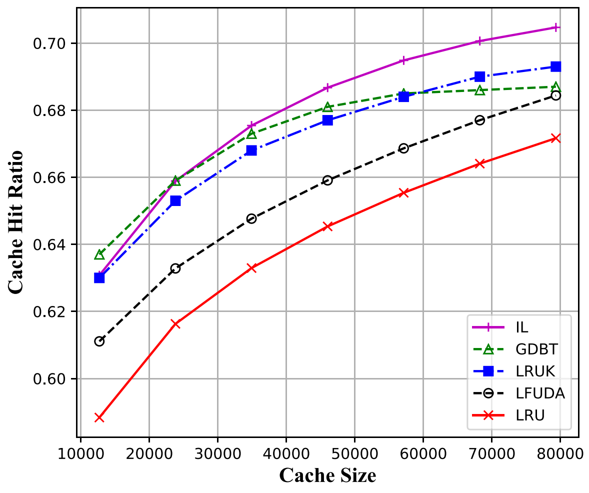

FIGURE 4. Hit-ratio comparison under different caching algorithms driven by Wikipedia trace (IL with the default 6 gaps as feature).

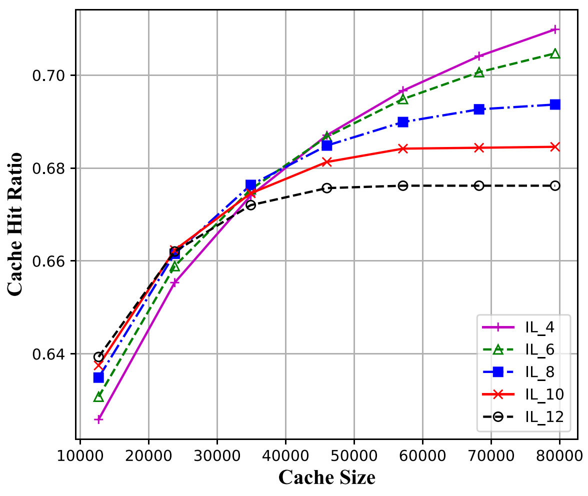

FIGURE 6. Hit-ratios of IL algorithm with different features driven by Wikipedia trace.

when cache size is larger than $34K$ and up to 3.7% higher with the $113K$ cache size. When the cache size is about $18K$, GDBT performs 3.2% better than IL. As for Wikipedia trace, IL algorithm commonly performs better than GDBT with the max advantage up to 2.6%. Except when cache size is about $12K$, GDBT performs 0.9% better than IL. The Wikipedia trace has a larger proportion of identical objects to the requests than CDN trace. Thus, requests for same objects are more than CDN trace, resulting higher hit ratio. Besides, each GDBT model is trained with traces up to 10× than each IL model. Thus, GDBT model is more competent at learning features of long term popular objects and is more competent with smaller cache. Because when cache size is getting larger, the cache hit ratio is raised by admitting medium popular objects. The GDBT model tends to not admit in medium popular objects for the impact of training with longer traces. However, IL based caching policy is more likely to cache medium popular objects than GDBT because of the shorter model updating period. To sum up, GDBT based caching policy performs better when the cache size is small, but IL based caching algorithm performs better when cache size is getting larger.

As shown in Fig. 5 and Fig. 6, we compare the impact for the cache hit ratio under IL algorithms with different features. The quantity of features plays an important role for caching performances. IL_12 denotes timestamp gap features $(Gap_1, \cdots, Gap_{12})$. With a larger time scope considered, the most popular objects might be requested several times more than the medium popular ones. Thus, the feature vectors differ a lot between the medium popular objects and the most popular ones. When cache size is small, large timestamp gaps such as Gap12 can help admit the most popular objects. However, when cache size is getting larger, medium popular objects might not be admitted into the cache for their feature Gap_{12} just being empty. IL_4 denotes features from $(Gap_1, \cdots, Gap_4)$. With less features considered in IL_4, the feature vectors of medium popular objects are more close to the most popular ones.

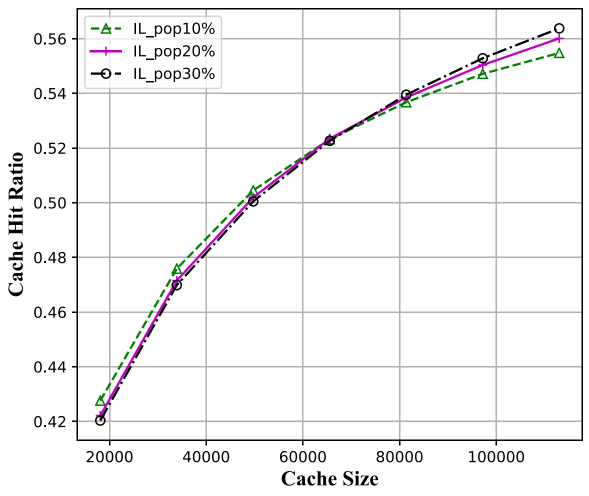

FIGURE 7. Hit-ratio comparison under different popularity label with CDN trace.

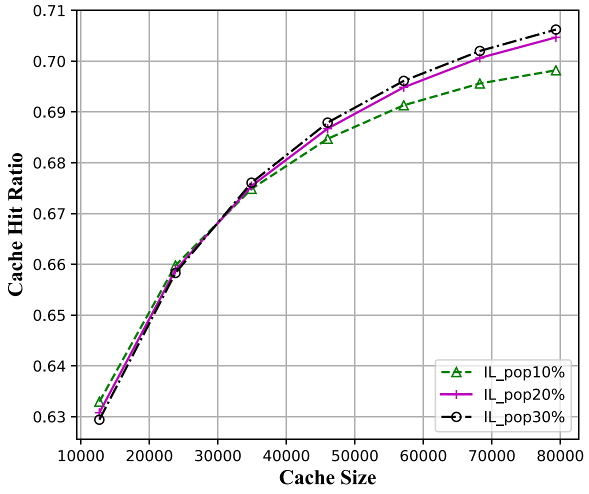

FIGURE 8. Hit-ratio comparison under different popularity label with Wikipedia trace.

The influence of popularity label for the cache hit ratio is illustrated in Fig. 7 and Fig. 8. According to Equation (5), objects within the top-K frequency in a time slot are taken as popularity labels $y_i^t = 1$. Different percentages from 10% to 30% are denoted as IL_pop10%, IL_pop20%, IL_pop30%, respectively. The strategy taking top-10% frequency as popularity label tends to admit the most popular objects and performs better when cache size is small. Only objects with features of top-10% popularity will be admitted into the cache. The medium popular objects are more likely to be admitted under the strategy taking top-30% as popular. Thus, the strategy taking top-30% as popular performs better when cache size is getting larger.

Fig. 9. shows the time consumption in dealing with user requests under different features. Since only time consumption of the learning algorithms is taken into consideration, different datasets should have same time cost under same algorithm. For each 1 million requests, GDBT based offline caching algorithm generates only one model, whereas our IL based algorithm generates up to 10 models to fit latest access patterns and achieve higher cache hit ratio. Under the default setting as features ($Gap_1 \cdots Gap6$), each model training time of IL_6 consumes only 3.6% of the model training time of GDBT. It uses 20 base predictors (the naive Bayes) to keep previous knowledge and thus costs more time on predictions. Taking both prediction and model updating into consideration, the overall throughput of IL_6 is about 24% higher than that of GDBT. Under different features, the time consumption for IL is always less than that of GDBT.

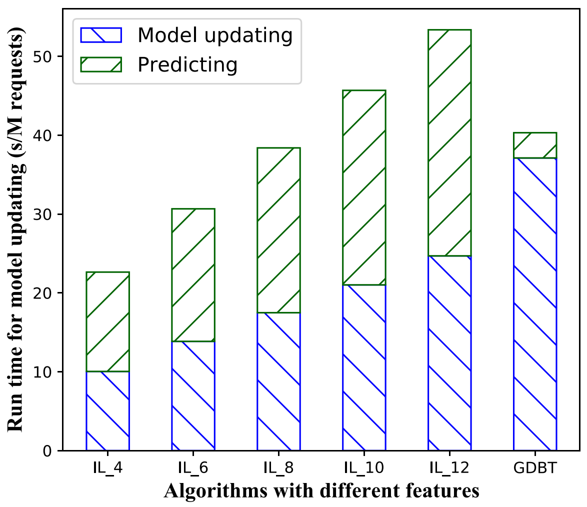

FIGURE 9. Time consumption comparison under different features.

VII. CONCLUSION

In this paper, we consider an edge caching mechanism for edge networks to deliver contents to end users. The challenge comes from the dynamic content popularity distribution and the frequent changed request patterns. By integrating the advances in edge networks with the advances in machine learning, the future role of edge applications is becoming more intelligent. It's expected that the performance of edge caching can be significantly improved by various learning algorithms. However, some learning algorithms for edge caching problems need to rebuild a new model periodically to adapt to changes of the access pattern of user requests, and then the knowledge learned from the past may be discarded. Besides, each model updating needs massive training data, during which the model might be outdated. This work presented an incremental learning based edge caching framework to preserve valuable knowledge and adapt to dynamic workloads. We implemented our incremental learning based cache prototype and evaluate its performance with various real-world workloads, and compared its performance with the state-of-the-art algorithms. The experimental results show that our proposed method performs better with real-world workloads in most cases. The extensions of our proposed framework for more general edge caching system models will be our on-going research topics.

REFERENCES

[1] C. V. Forecast, "Cisco visual networking index: Forecast and trends, 2017–2022," Cisco, San Jose, CA, USA, White Paper 1551296909190103, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html
[2] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caireieee, "The role of caching in future communication systems and networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1111–1125, Jun. 2018.
[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.
[4] S. Park and C. Park, "FRD: A filtering based buffer cache algorithm that considers both frequency and reuse distance," in *Proc. 33rd Int. Conf. Massive Storage Syst. Technol. (MSST)*, 2017.
[5] G. Einziger, R. Friedman, and B. Manes, "TinyLFU: A highly efficient cache admission policy," *ACM Trans. Storage*, vol. 13, no. 4, p. 35, 2017.
[6] M. Z. Shafiq, A. X. Liu, and A. R. Khakpour, "Revisiting caching in content delivery networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 567–568, Jun. 2014.
[7] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5G wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 1617–1655, Feb. 2016.
[8] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "Making content caching policies 'smart' using the deepcache framework," *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 5, pp. 64–69, Jan. 2019.
[9] D. S. Berger, "Towards lightweight and robust machine learning for CDN caching," in *Proc. 17th ACM Workshop Hot Topics Netw. (HotNets)*, 2018, pp. 134–140.
[10] H. H. Liu, R. Viswanathan, M. Calder, A. Akella, R. Mahajan, J. Padhye, and M. Zhang, "Efficiently delivering online services over integrated infrastructure," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 77–90.
[11] V. Losing, B. Hammer, and H. Wersing, "Incremental on-line learning: A review and comparison of state of the art algorithms," *Neurocomputing*, vol. 275, pp. 1261–1274, Jan. 2018.
[12] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.
[13] E. Zeydan, E. Bastug, M. Bennis, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data caching for networking: Moving from cloud to edge," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 36–42, Sep. 2016.
[14] S. Li, J. Xu, M. Van Der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[15] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "AdaptSize: Orchestrating the hot object memory cache in a content delivery network," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 483–498.
[16] N. Beckmann, H. Chen, and A. Cidon, "LHD: Improving cache hit rate by maximizing hit density," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 389–403.
[17] Y. Zeng, Y. Huang, Z. Liu, and Y. Yang, "Joint online edge caching and load balancing for mobile data offloading in 5G networks," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 923–933.
[18] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 180–190, Feb. 2018.
[19] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Proc. 52nd Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2018, pp. 1–6.
[20] S. Pei, T. Shen, X. Wang, C. Gu, Z. Ning, X. Ye, and N. Xiong, "3DACN: 3D augmented convolutional network for time series data," *Inf. Sci.*, vol. 513, pp. 17–29, Mar. 2020.
[21] A. Sadeghi, G. Wang, and G. B. Giannakis, "Deep reinforcement learning for adaptive caching in hierarchical content delivery networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 4, pp. 1024–1033, Dec. 2019.
[22] J. Gama, I. Žliobaitÿ, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *CSURACM Comput. Surv.*, vol. 46, no. 4, pp. 1–37, Mar. 2014.
[23] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Trans. Knowl. Data Eng.*, to be published.
[24] Y. Ji, T. Shen, S. Pei, and H. Liu, "Migration mechanism of heterogeneous memory pages using a two-way Hash chain list," *Sci. Sinica-Inf.*, vol. 49, no. 9, pp. 1138–1158, Sep. 2019.
[25] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, Nov. 2015.
[26] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, May 2010.
[27] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," *SIGMOD Rec.*, vol. 22, no. 2, pp. 297–306, Jun. 1993.
[28] J. Dilley, M. Arlitt, and S. Perret, "Enhancement and validation of squid's cache replacement policy," HP Lab., Palo Alto, CA, USA, Tech. Rep. HPL 69, 1999.
[29] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Comput. Netw.*, vol. 53, no. 11, pp. 1830–1845, Jul. 2009.

GUANGPING XU (Member, IEEE) received the B.S. degree in computer science from the Tianjin University of Technology, in 2000, and the M.S. degree in computer science and the Ph.D. degree in computer science from Nankai University, China, in 2005 and 2009, respectively. He is currently an Associate Professor with the Tianjin University of Technology. His research interests include distributed storage systems and algorithm optimization.

BO TANG received the B.S. degree in electronic and information engineering from the China University of Geosciences, Wuhan, China, in 2018. He is currently pursuing the M.S. degree in computer science from the Tianjin University of Technology. He has coauthored several articles in international conferences. His research interests are in distributed systems, storage systems, information theory, and edge computing.

LIMING YUAN received the Ph.D. degree in computer science and technology from the Harbin Institute of Technology, China, in 2014. He is currently working as a Lecturer with the School of Computer Science and Engineering, Tianjin University of Technology, China. His research interests are mainly in machine learning and multimedia processing.

YANBING XUE received the B.S. degree in computer science from Shandong Normal University, Jinan, China, in 2002, and the M.S. degree in computer application from the Tianjin University of Technology, Tianjin, China, in 2005. He is currently an Associate Professor with the Department of Computer Science and Engineering, Tianjin University of Technology. His research interests include computer multimedia and machine learning.

ZAN GAO received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2011. Since 2011, he has been working with the Key Laboratory of Computer Vision and System, School of Computer Science and Engineering, Ministry of Education, Tianjin University of Technology. His research interests include artificial intelligence, multimedia analysis and retrieval, and machine learning.

SALWA MOSTAFA received the B.S. and M.S. degrees in electrical and communication engineering from the Faculty of Electronic Engineering, Menofia University, Menofia, Egypt, in 2012 and 2015, respectively. She is currently pursuing the Ph.D. degree in electrical engineering with the City University of Hong Kong. Her main research interests include edge caching and computing, coded caching, non-orthogonal multiple access, network information theory, coding theory, algorithms, and computational complexity.

CHI WAN SUNG (Senior Member, IEEE) received the B.Eng., M.Phil., and Ph.D. degrees in information engineering from The Chinese University of Hong Kong, in 1993, 1995, and 1998, respectively. After graduation, he was an Assistant Professor at The Chinese University of Hong Kong. He joined the Faculty at the City University of Hong Kong in 2000, where he is currently an Associate Professor with the Department of Electronic Engineering. His research interest is on coding, communications, and networking, with emphasis on algorithm design and complexity analysis. He was an Associate Editor of the *Transactions on Emerging Telecommunications Technologies* from 2013 to 2016. He is currently on the Editorial Boards of the *ETRI Journal* and *Electronics Letters*.