Smart Data-Driven Proactive Push to Edge Network for User-Generated Videos

Xiaoteng Ma*†, Qing Li*, Junkun Peng*‡, Gareth Tyson§, Ziwen Ye‡, Shisong Tang‡, Qian Ma¶, Shengbin Meng¶, Gabriel-Miro Muntean‖
*Peng Cheng Laboratory, China †Tsinghua-Berkeley Shenzhen Institute, Tsinghua University, China
‡Tsinghua Shenzhen International Graduate School, Tsinghua University, China
§IoT Thrust, Hong Kong University of Science and Technology (GZ), China
¶ByteDance Inc., China ‖School of Electronic Engineering, Dublin City University, Ireland
Emails: †‡{maxt17, pjk20, yezw21, tangss}@mails.tsinghua.edu.cn, *liq@pcl.ac.cn, §gtyson@ust.hk
¶{maqian.zero, mengshengbin}@bytedance.com, ‖gabriel.muntean@dcu.ie

Abstract*—To reduce costs and improve performance, video Content Delivery Networks (CDNs) have started to incorporate lightweight edge nodes, *e.g.*, WiFi access points. Because of this, it is necessary for CDNs to intelligently select which video files should be placed at their core data centers vs. these edge nodes. This is more complex than traditional CDN management, as lightweight edge nodes are much more numerous and unstable than data centers. With this in mind, we present SDPush —- a system for managing content placement in edge CDNs. SDPush tackles two problems. First, it is necessary for SDPush to select *which files* to proactive push. To address this, we build a file popularity prediction model that effectively identifies video files that will receive many views. Second, SDPush should determine *how many replicas* of each file to push. To address this, we design a model to predict the benefits of pushing particular files (regarding traffic savings) and then formulate the replica decision problem as a lightweight problem, which is solvable within seconds, even for platforms that accommodate millions of daily active users. Through a trace-driven evaluation and a live deployment on a real video platform, we validate SDPush's effectiveness, offloading peak-period traffic by 12.1% to 23.9% from the data center to edge nodes, thereby reducing the CDN costs.

***Index Terms*—Data-Driven, Popularity Prediction, Proactive Push, Lightweight Edge Caches**

I. Introduction

Content **D**elivery **N**etworks (CDNs) rely heavily on centralized data centers and dedicated racks, often placed in **I**nternet **S**ervice **P**roviders (ISPs). To improve performance and reduce costs, some CDNs (*e.g.*, Aliyun [1] and Lumen [2]) have started to extend cache capacity to edge networks using ultra-lightweight devices such as WiFi access points, Femto Base Stations, and Internet of Things devices. These devices, often leased from other companies (*e.g.*, Aliyun [1], PacketFabric [3]), effectively supplement the more conventional dedicated infrastructure by caching content.

This new model of operation offers several benefits. These ultra-lightweight edge nodes have lower bandwidth prices [4] and are far more numerous. By combining the high availability of their data centers with the proximity and low cost of the edge nodes, CDNs can reduce operating costs by redirecting clients' requests to nearby heir edge locations [1], [5]. Reducing costs can also reduce the price of its services, making it more commercially competitive. This does introduce a number of challenges, though, as these ultra-lightweight edge deployments often have scarce resources. Due to this, it is vital to ensure that each edge node has the files demanded in its region, especially during peak periods (*i.e.*, 20:00 $\sim$ 21:00). From our measurements taken from Xigua, one of the top user-generated video content providers in China, we find that the request distribution of files peaks between 1.66x – 2.44x higher than the average. Thus, incorrect file caching leads to a surge in requests to the data center's backend, driving up CDN costs. This is particularly critical since both network operators and content providers usually get paid based on peak periods (rather than average), as seen in iQiYi [6] and Kwai [7].

Corresponding author: Qing Li. Email: liq@pcl.ac.cn

To overcome this, several systems have been proposed to proactively *push* popular content to edge devices (*e.g.*, during low traffic periods) [6], [8], [9]. These previous schemes have several limitations. They solely rely on video-level attributes to predict popularity. Yet our later analysis shows that features related to encoding can also be highly predictive (Section II-B). Also, they use regression-based methods that accurately predict highly popular files but exhibit limited effectiveness in predicting files with moderate or low popularity [10]. This poses a significant challenge for SDPush to select files for proactive push accurately. Additionally, these schemes solely rely on a single feature of the file (*e.g.*, popularity or size) to determine the number of edge nodes to which the file should be pushed, without analyzing the impact of these factors on data center bandwidth savings and integrating them together. This could result in pushing the wrong file and cache wastage. We argue that, by overcoming these prior limitations, we can better place video files at the ultra-lightweight edge nods before they are requested, such that traffic can be localized, and offloaded from the (expensive) data centers. We distill this task down to overcoming two key challenges: (*i*) determining which files to push to the edge nodes, and (*ii*) how many edge nodes to push each file to.

To address the above challenges, this paper introduces the **S**mart **D**ata-driven Proactive **Push** mechanism (SDPush). The goal of SDPush is to reduce the peak traffic load on the core data centers. It does this by proactively pushing file replicas to lightweight edge nodes, so that more requests can be handled at the edge. To underpin and then evaluate our design, we build a dataset with the help of a leading professional user-generated video platform, encompassing 200 million client traces and 2.5 million file records (Section II-A). This dataset reveals issues that previous studies have not addressed, including the high correlation between encoding features and file popularity (Section II-B), as well as factors influencing the optimal number of edge nodes at which each file should be cached (Section II-C). We rely on these insights to overcome the challenges of the proactive push mechanism.

The **first challenge** for SDPush is determining *which* files to push to the edge nodes. For this, accurate popularity prediction is crucial. Previous solutions [6], [8] work well in predicting the volume of requests for popular files. However, it has been shown that these previous models perform poorly for moderate and low-popularity files [10]. Consequently, we adopt a pairwise-based popularity ranking, which discriminates between moderate and low-popularity files. Our approach compensates for inaccuracies in the predictions for moderate and low-popularity files (Section III-C).

The **second challenge** involves determining *how many* edge nodes to push each file to. Pushing too many replicas of a given file will deprive other files of cache resources. Conversely, underallocation may hinder performance. Thus, we must identify the optimal number of edge nodes for caching each file in real-time, a daunting task for systems that must push millions of replicas within short periods. To address this, we model the non-linear correlation between the number of edge caches per file and the potential data center's bandwidth savings. Using our unique dataset, we develop a model that predicts the total traffic from all edge nodes for a given file under different cache conditions (Section III-D). We then formulate the problem as an integer programming problem, and show how we can scale-up our implementation to calculate results for millions of files in seconds (Section III-E).

The contributions of this paper are as follows:

- We conduct an extensive study on video file access patterns using 200 million client traces. We identify file features that effectively predict popularity and show how this can be used for improving proactive push.
- We introduce SDPush, a smart data-driven proactive push mechanism that reduces peak-period data center traffic by identifying and pushing high-traffic files during peak periods to the edge.
- We present a trace-driven evaluation and real-world deployment of SDPush. Our real-world deployment shows that SDPush effectively offloads peak period traffic by between 12.1% to 23.9% from the data center to edge nodes, compared to the previous mechanisms.

II. Motivation

A. Dataset

Our motivation relies on a unique dataset taken from Xigua, a top professional user-generated video platform in China. The dataset consists of three parts. The first part is the *client request data*, which contain about 200 million client traces that span two weeks in November 2021. The key features include access time, leave time, client ID, file ID, connected edge node ID, and download size from each edge node and data center, respectively, per video view. Each file is encoded using specific encoding parameters (*i.e.*, resolution, codec, and quality), with a duration of 1 to 4 minutes. The second part is the *file information data*, which contains 2.5 million file records that appeared in the first dataset's time range. It includes file resolution, codec, quality, size, content author, category, and publish time. The third part is the *author information data*. This covers their number of fans, published videos, and total received likes (appearing in the second dataset).

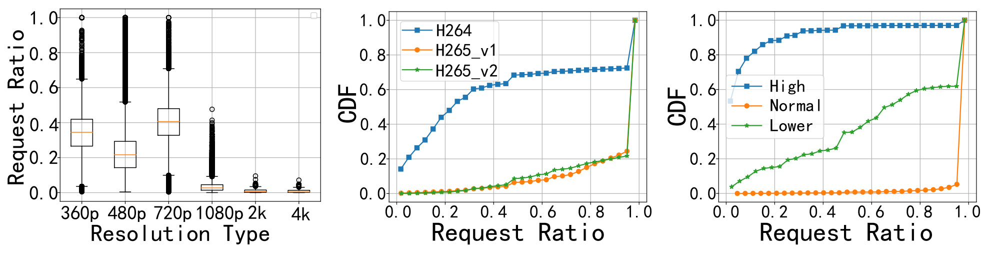
B. Impact of File Features on Popularity

We first explore how different file features influence popularity.

1) Encoding features: Clients are more likely to request files with a resolution below 720p. Fig. 1(a) shows the request ratio of different resolution files. The request ratio refers to the ratio of requests for files of each resolution vs. requests for all files at the peak period (*i.e.*, 20:00∼21:00) in a day. We see that 360p, 480p, and 720p have a high probability of having requests during the peak period. Therefore, SDPush should preferentially push files with these resolutions.

Clients prefer to request H265 encoded files with a "Normal" quality level. Fig. 1(b) and Fig. 1(c) show the request ratio for different codecs and quality levels. The "quality" indicator represents three different encoding parameters (*i.e.*, constant rate factor distribution) for a video under a certain resolution. We see that popularity varies significantly across different codecs and quality levels. This is because when receiving an original video, the data center server first encodes the video file in a low-computational manner (*i.e.*, H264 coding). Such encoded files usually have large sizes and relatively low quality. If the file becomes popular, the data center utilizes a more computationally expensive encoding method (*i.e.*, H265_v1 or H265_v2). The newly encoded file has a smaller size and higher quality so that clients who request such files experience a better quality of experience. When files using the new codec are produced, clients who support playing these newly encoded files prefer to request files with the new codec, thus reducing the popularity of old codec files. Although the specific codes and encoding rates may vary in the future, we argue that the observation remains the same: *a subset of codecs and resolutions will dominate at any given period.*

2) Social Features: Videos published by a popular author are more likely to become popular. Fig. 2 shows the relationship between files' peak-period request volume and the video author's number of fans/likes. We utilize min-max normalization to normalize the values on the Y-axis for each metric. Unsurprisingly, the number of peak-period requests increases as the authors' number of fans and the average



(a) Resolution (b) Codec (c) Quality

Fig. 1: Encoding Features

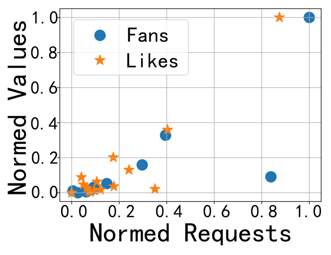
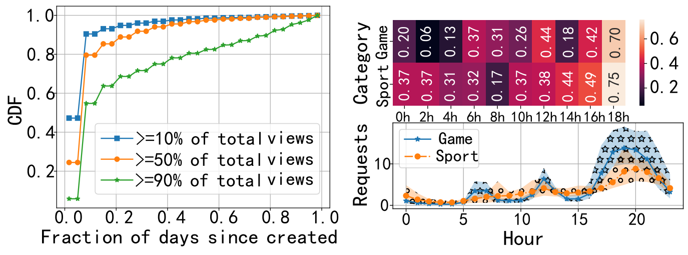


Fig. 2: Social Features



(a) Age (b) Category

Fig. 3: Request Pattern Features

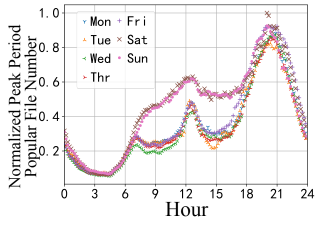


Fig. 4: # of Popular Files

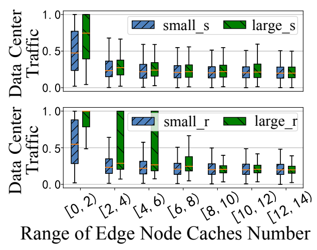


Fig. 5: Benefit Heterogeneity

number of received likes increase. Thus, *the author's fans/likes may be useful for making popularity predictions.*

3) Request Pattern Features: Most of the requests arrive early in the file's lifecycle. Fig. 3(a) presents the fraction of days required to get $n\%$ of each file's view count. This covers up to two weeks after video generation. 63% of the files get more than 90% views in the first 20% of their lifetime. This suggests that most files should only be pushed early in their lifecycle.

We also observe that such differences may vary across video categories. To study this, Fig. 3(b) shows a heatmap of the Pearson correlation between the *average* request number in each daily period and the daily *peak* (just for weekdays). It also presents the average number of requests per hour. We show results for all videos in two example video categories. For gaming videos, only the request volume in the early morning, noon, and evening correlate with the request volume in the peak period (*i.e.*, the Pearson coefficient is larger than 0.3). In contrast, for sports videos, the volume of requests in most periods correlates well with the number of requests in the peak period. This trend probably emerges due to the differences among audiences, *e.g.*, most viewers of games are young people who perhaps watch videos during their breaks [11]. The result highlights that *per-category temporal features might be useful when making popularity predictions for different videos.*

4) Temporal Features: The number of files that should be pushed to the edge network varies at different times of the day. Any content push mechanism should be executed within a given interval, T. For example, with fixed time intervals, if we decide at time t_d, the next decision should be made at time t_d+T. We call the period [t_d, t_d+T) a "push period".

Fig. 4 presents the normalized number of the peak-period popular files during each push period according to the data from November 7th to November 13th, where we set T to 10 minutes. The peak-period popular file number can be calculated as $\sum_{j\in\vec{\mathbf{J}_d}} \mathbb{1}_{\{r_j^p|r_j^p\geq C\}}(r_j^p)x_j^{t_d}$, where $\mathbb{1}_{\{r_j^p|r_j^p\geq C\}}(r_j^p) = \begin{cases} 1 & r_j^p \geq C \\ 0 & r_j^p < C \end{cases}$, r_j^p is the requests volume of file j during the peak period, C is the cut-off threshold of popular and unpopular files, and $\vec{\mathbf{J}_d}$ is the set of selected files' ID at time t_d. For Fig. 4, we set C=50 and $\vec{\mathbf{J}_d}$ includes the ID of files that have been requested in the time period [t_d-3600, t_d). We use min-max normalization to normalize the value of peak-period popular file numbers according to the maximum and minimum values in the whole dataset.

Fig. 4 reveals there is a large difference in the number of popular files at peak periods in different push periods. For instance, the number of popular files at 12:00 is 9.3$\sim$12.5 times the number of popular files at 03:00. This indicates that *we should vary the total number of files pushed in different periods to avoid deploying unpopular files that may occupy cache resources of popular files.*

C. Impact of Number of Edge Node Caches

The above study has identified the file features that correlate with popularity. Next, we explore the individual bandwidth savings that can be achieved when pushing content to edge nodes. We conjecture that when different files are deployed to the same number of edge nodes, there may be differences in the benefits attained (because of heterogeneity in file size and popularity). Fig. 5 shows the distribution of the normalized data center traffic attained per file in different ranges of edge nodes that cache a certain file during the peak period. For each file, the traffic of the edge nodes plus the traffic of the data center equals the total traffic attained by the file. To understand how this varies across types of files, we separate files into four categories: "small_s" and "large_s" indicates files with a small size (0MB$\sim$500MB) and large size (500MB$\sim$1000MB), while "small_r" and "large_r" represents files with a small number of requests (0$\sim$50) and a large number of requests (larger than 50), respectively.

Fig. 5 illustrates that larger files and those with more requests require more edge nodes due to longer session durations and simultaneous request limitations, underscoring the need for *a proactive push mechanism that adapts the number of replicas on a per-file basis.*

III. SDPush Design

A. Problem Formulation

We first define the problem SDPush tackles. Assuming that SDPush processes J files daily and each push period lasts T minutes, we formulate the objective of SDPush as follows:

$$\arg\max_{\mathscr{D}} \sum_{t_d \in \vec{\mathbf{t}}} \sum_{j=1}^{J} g_j^{t_d} \times \varphi(g_j^{t_d}, r_j^p, s_j)$$
$$s.t. \begin{cases} g_j^{t_d} = g_j^{t_d - T} + \rho_j^{t_d - T} - e_j^{[t_d - T, t_d)} \\ \sum_{j=1}^{J} \rho_j^{t_d} \le \omega^{t_d}, \rho_j \in Z_+ \cup \{0\} \\ \sum_{d=1}^{D} \omega^{t_d} \le \Gamma, \forall t_d \in \vec{\mathbf{t}}, \forall j \in [1, J] \end{cases} \tag{1}$$

where $\varphi(\cdot)$ is the model to predict the average traffic attained for all streams for a given file from all edge nodes during the peak period (described in Section III-D). Note that file segments can be downloaded from the edge or data center nodes during a video view. A higher traffic from the edge network implies a higher utilization of edge nodes, and therefore more data center bandwidth savings. As bandwidth costs at the edge are cheaper, offloading from the data center is more desirable [1], [5]. r_j^p is the volume of requests for the file j at the peak period. s_j is the size of file j. $\vec{\mathbf{t}} = \{t_1, t_2, ..., t_d, ..., t_D\}$ is the start time of the push period. $g_j^{t_d}$ is the number of replicas for file j cached at the edge network in the push period starting at time t_d. Note that each edge node caches at most one replica of a particular file. $\rho_j^{t_d}$ is the number of replicas of file j that decided to be pushed in the push period starting at time t_d. $\vec{D^{t_d}}$=$\{\rho_1^{t_d}$, $\rho_2^{t_d}$, ..., $\rho_j^{t_d}$, ...$\}$ is the set of ρ^{t_d} for each file. $\mathscr{D} = \{\vec{D^{t_1}}, \vec{D^{t_2}}, ...\}$ is the set of $\vec{D^{t_d}}$. ω^{t_d} is the upper bound of the total number of replicas pushed in the push period starting at time t_d. e_j is the number of evicted replicas of file, j, in the time period $[t_d - T, t_d)$. Γ is the maximum number of pushed replicas in a day, which is related to the number of edge nodes and the storage capacity.

To solve the problem in Eq.(1), we must select D^{t_d} at time t_d. We formulate the problem's objective at time t_d as follows:

$$\arg\max_{D^{t_d}} \sum_{j=1}^{J} [(\rho_j^{t_d} + g_j^{t_d}) \times \varphi(\rho_j^{t_d} + g_j^{t_d}, r_j^{t_d}, s_j) - g_j^{t_d} \times \varphi(g_j^{t_d}, r_j^{t_d}, s_j)]$$
$$s.t. \begin{cases} g_j^{t_d} = g_j^{t_d - T} + \rho_j^{t_d - T} - e_j^{[t_d - T, t_d)} \\ \sum_{j=1}^{J} \rho_j^{t_d} \le \omega^{t_d}; x_j^{t_d} \rho_j^{t_d} = 0 \\ \rho_j^{t_d} \in Z_+ \cup \{0\}; x_j^{t_d} \in \{0, 1\}; \forall j \in [1, J] \end{cases} \tag{2}$$

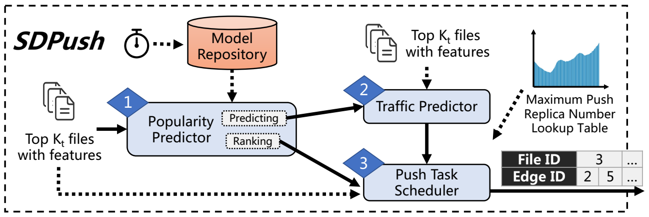


Fig. 6: SDPush Workflow

where $r_j^{t_d}$ is the predicted number of requests per file, j, at time t_d. $x_j^{t_d}$ is a binary variable: $x_j^{t_d} = 0$ when the file j is selected at time t_d; otherwise $x_j^{t_d} = 1$.

It is worth noting that motivated by Fig. 4 and the analysis in Section II-B, varying ω^{t_d} according to the push period has the potential to avoid replicating too many unpopular files. Thus, we should predict the number of popular files to push in different push periods to achieve the above goals. Fortunately, predicting this distribution is not a difficult task. The ratio of the peak-period popular files at each push period is quite stable on different weekdays and weekends (see Fig. 4). This indicates that we can store the upper bound ω^{t_d} for the number of push replicas during different push periods of weekdays and weekends in a lookup table, respectively.

B. SDPush Overview

Fig. 6 illustrates the SDPush workflow during a push period. The solid line represents the working process, and the dotted line indicates the transmission of information. The SDPush architecture comprises the following three blocks: the Popularity Predictor, the Traffic Predictor, and the Push Task Scheduler. These three blocks run sequentially to complete the whole workflow of SDPush.

Step 1: Popularity Prediction (Section III-C): In Step 1, SDPush sorts the files according to the requested volume of files in the past H hours. It then extracts the top K_t files with fewer than three replicas at the edge as the potential files to push. The Popularity Predictor ranks and predicts the selected files' peak-period request volume based on its prediction models. The Popularity Predictor may use different models in different push periods.

Step 2: Traffic Prediction (Section III-D): In Step 2, the Traffic Predictor utilizes the file size and the predicted peak-period request volume to predict the traffic from edge nodes under different edge node caches.

Step 3: Push Task Scheduling (Section III-E): In Step 3, the Push Task Scheduler decides the number of replicas to push for files according to the Traffic Predictor's output and a lookup table of the maximum number of pushed replicas per push period. Finally, our edge allocator determines the target edge node where each replica is pushed.

C. Step 1: Popularity Prediction

There are two requirements for the Popularity Predictor. For selecting *which* files to push, the Popularity Predictor should calculate the priority rank of each file. For deciding *how many* replicas to push, the Popularity Predictor should then give an

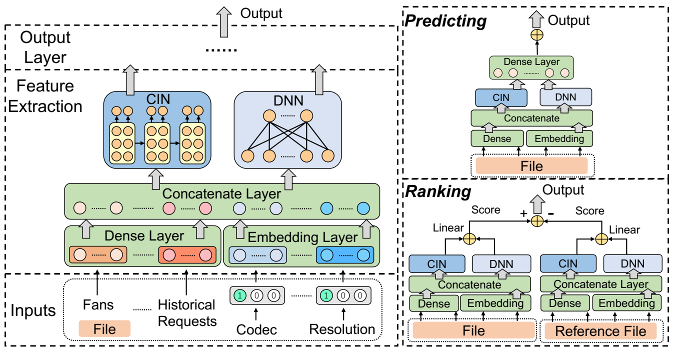


Fig. 7: Popularity Prediction Model

accurate estimate of the number of requests for files (during the peak period). Because optimizing one of the indicators will not lead to the decline of the other, we use different prediction models to predict the popularity ranking and the number of requests per file, respectively. Both models have identical input formats and feature extraction modules, as shown in Fig. 7. Only the output layer is different.

Based on the analysis in Section II-B, we find that the popularity of a file is affected by various features. Therefore, we next explore and utilize the relationship between multi-dimensional features and file popularity. We try a variety of feature fusion schemes and find that the popularity prediction architecture that attains the best performance combines the explicit features extracted by a Compressed Interaction Network (CIN), the implicit features extracted by a Deep Neural Network (DNN), and the linear combination features extracted via Linear Regression (LR). Such a method is inspired by the xDeepFM [12] model, which was designed for click-through rate prediction tasks. We do not show the linear regression part in Fig. 7 for the sake of brevity. The categorical features (*i.e.*, codec, resolution, quality, and video category) are encoded as one-hot vectors and then embedded. After embedding, the neural network expands continuous variables (*i.e.*, historical requests, file age, author fans, and received likes) to the same dimensions as categorical features. Then we concatenate the dense and embedding layers and connect them to the compressed interaction and deep neural networks. The ranking model output is a score representing the popularity difference between the file that needs to be predicted and the reference file. The loss function is $\mathcal{L}_r = -\frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}log\hat{y_{ij}} + (1-y_{ij})log(1-\hat{y_{ij}})$, where $\hat{y_{ij}} = \frac{e^{o_{ij}}}{1+e^{o_{ij}}}$. o_{ij} is the output of the model and could be any real number. If o_{ij} is positive and the value is higher, the popularity of file i is higher than that of file j. Conversely, if o_{ij} is negative and the value is lower, file j is less popular than file i. $y_{ij} \in \{0, 0.5, 1\}$ and represents the real relative popularity relationship between file i and file j. The values of y are 1, 0.5, and 0, respectively, representing that file i is more popular than file j, file i and file j have the same popularity, and file j is more popular than file i. M and N refer to the number of references and predicted files, respectively. To adapt to daily changes in the peak popularity distribution of files, we re-select Q reference files daily when using the model. We have experimented with various Q values and found that when $Q > 5$, the ranking performance does not change with the increase of Q. So, we set Q=5. The reference files used for the current day are those ranked around the 70th percentile regarding requests number during the last day's peak periods, sorted in descending order. The reason why we choose files with relatively low popularity is that these files' popularity is not easily distinguishable. We also try different files as reference files, and find that using files ranked by popularity between 60% and 80% performs best in ranking.

TABLE I: Performance of the popularity prediction models

Metrics	Predicting MSE(1e-6)	Predicting R^2	Ranking NDCG	Ranking AUC
LR	11.2	0.6093	0.8742	0.9058
XGBoost	9.25	0.6772	0.9117	0.9318
DNN [6]	8.99	0.6842	0.9189	0.9446
DeepFM [13]	8.81	0.6865	0.9219	0.9467
DCN [14]	8.72	0.6902	0.9210	0.9480
xDeepFM [12]	**8.56**	**0.6941**	**0.9246**	**0.9501**

TABLE II: Performance of the traffic prediction models

	LR	DT	RF	GBDT	DNN	XGBoost
MSE(1e-6)	11.87	9.54	6.82	5.61	5.54	**5.23**
R^2	0.62	0.68	0.79	0.81	0.82	**0.83**

For the popularity prediction model, the output approximates the volume of file request in the peak period. The loss function is expressed as $\mathcal{L}_p = \sum_{i=1}^{N}(\hat{r_i} - r_i)^2$, where $\hat{r_i}$ and r_i are the file's predicted and real request volume, respectively.

For context, Table I summarizes the performance of various schemes for fusing multidimensional features. For the ranking task, we use **N**ormalized **D**iscounted **C**umulative **G**ain (NDCG) and **A**era **U**nder the **C**urve (AUC) to evaluate its ranking accuracy. NDCG evaluates files whose actual popularity ranks in the top 50%. AUC is used to judge whether the comparison between the prediction result and the number of requests in the peak period for the reference file is correct. For the peak-period request volume prediction task, we evaluate its prediction accuracy by Mean Squared Error (MSE) and R-square (R^2). The result implies that the feature extraction method used by xDeepFM makes the best use of the features.

D. Step 2: Traffic Prediction

The next step is establishing and using the model $\varphi(\cdot)$ mentioned in Section III-A, which predicts the average traffic attained for all streams for a given file from all edge nodes during the peak period.

We use the dataset described in Section II-A to train the model $\varphi(\cdot)$. To train the proposed model, we compile a feature set containing the file size, number of file replicas in the network in the peak period, total number of requests in the peak period for the file, and total traffic that clients obtain when accessing the file from the edge nodes during the peak period. Then the peak-period traffic w_j from an edge node for a file j is calculated as $\varphi(g_j, r_j, s_j)$, where g_j is the number of edge caches of file j at the edge network, r_j is the number of requests for video file j in the peak period, and s_j is the size of the file j. To train $\varphi(\cdot)$, we experiment with multiple models: Linear Regression (LR), Decision Tree (DT), Random Forest (RF), Gradient-Boosted Decision Trees (GBDT), Deep Neural Network (DNN), and eXtreme Gradient Boosting (XGBoost). Our results show XGBoost performs best based on the input features, as listed in Table II. Its R^2 between the estimated and real edge traffic is 0.83, which implies that it can accurately estimate the traffic from the edge network [15].

However, formulas cannot express $\varphi(\cdot)$ directly. This prevents us from obtaining an analytical solution for Eq. (2). We, therefore, fit the model, $\varphi(\cdot)$, using polynomial fitting to tackle this problem scalably. At each time t_d, the Push Task Scheduler obtains the selected files' predicted popularity and size. This allows us to estimate the traffic, $\phi_j(\rho)$, from an edge node serving the file, j, when we push ρ replicas to the edge. We find that there is a good fitting performance (0.9672 R^2 score) when the polynomial degree is equal to 4. Therefore, considering the complexity of the problem, we use the fourth-order fitting method to fit the relationship between the number of replicas to push, ρ, and the normalized edge traffic for a file, $\phi_j^{t_d}$, at time t_d. This can be expressed as $\phi_j^{t_d}(\rho) = \alpha_j^{t_d}\rho^4 + \beta_j^{t_d}\rho^3 + \gamma_j^{t_d}\rho^2 + \delta_j^{t_d}\rho + \eta_j^{t_d}$, where $\alpha_j^{t_d}$, $\beta_j^{t_d}$, $\gamma_j^{t_d}$, $\delta_j^{t_d}$ and $\eta_j^{t_d}$ are all constant terms which are related to $g_j^{t_d}$, $r_j^{t_d}$ and s_j. Finally, we formulate the objective of Eq.(2):

$$\begin{aligned} \arg\max_{D^{t_d}} \sum_{j=1}^{J} [a_j^{t_d}(\rho_j^{t_d})^5 + b_j^{t_d}(\rho_j^{t_d})^4 + c_j^{t_d}(\rho_j^{t_d})^3 \\ + d_j^{t_d}(\rho_j^{t_d})^2 + f_j^{t_d}\rho_j^{t_d}] \\ s.t.\ Constraints\ in\ Eq.(2) \end{aligned} \tag{3}$$

where $a_j^{t_d} = \alpha_j^{t_d}$, $b_j^{t_d} = 5\alpha_j^{t_d}g_j^{t_d} + \beta_j^{t_d}$, $c_j^{t_d} = 10\alpha_j^{t_d}(g_j^{t_d})^2 + 4\beta_j^{t_d}g_j^{t_d} + \gamma_j^{t_d}$, $d_j^{t_d} = 10\alpha_j^{t_d}(g_j^{t_d})^3 + 6\beta_j^{t_d}(g_j^{t_d})^2 + 3\gamma_j^{t_d}g_j^{t_d} + \delta_j^{t_d}$, $f_j^{t_d} = 5\alpha_j^{t_d}(g_j^{t_d})^4 + 4\beta_j^{t_d}(g_j^{t_d})^3 + 3\gamma_j^{t_d}(g_j^{t_d})^2 + 2\delta_j^{t_d}g_j^{t_d} + \eta_j^{t_d}$.

However, our experience has shown that there still remain scalability challenges when dealing with a large number of files. Fortunately, based on our data, we find that only the top 1% of files ranked by the peak period's request number require more than 30 edge nodes to store their replicas. Thus, we can reduce the complexity of solving this problem by limiting the maximum number of replicas to M (30 in our paper) for each file at every single push period (*i.e.*, $\rho_j^{t_d} \in [0, M]$, $\forall j \in [1, J]$). For files that are predicted to be in the top 1% of the request number of the peak period, the Push Task Scheduler selects them again in the next period. It then determines how many more replicas to push. Thus, we can convert Eq. (3) to a 0-1 integer programming problem with $O(MJ)$ complexity.

E. Step 3: Push Task Scheduling

In this section, we describe how the Push Task Scheduler uses the Popularity Predictor and thetraffic Predictor to make real-time decisions on which replicas to push to which edge nodes. The Push Task Scheduler process is divided into three sub-steps: (*i*) **File Clustering**, (*ii*) **Parallel Optimization**, and (*iii*) **Edge Allocation**, which are executed sequentially (see Fig. 8).

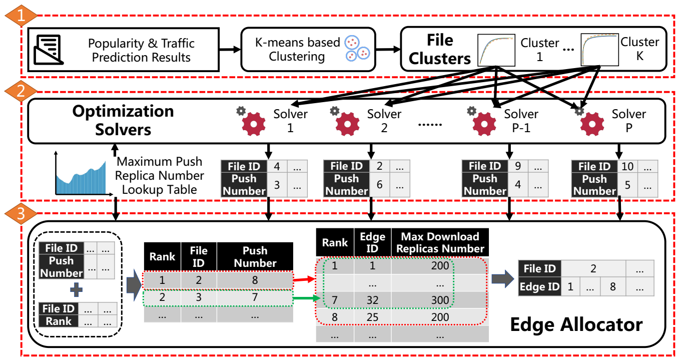


Fig. 8: Push Task Scheduler Workflow

Sub-step 3.1: File Clustering. At time t_d, the popularity and traffic prediction results are fed into the workflow. SDPush first uses K-means to cluster files with a similar relationship between the number of edge nodes and the average traffic attained for all streams for a given file from all edge nodes. This groups together files that should be treated similarly. The similarity among files is estimated by the Euclidean distance of the traffic prediction results (*i.e.*, $a_j^{t_d}$, $b_j^{t_d}$, $c_j^{t_d}$, $d_j^{t_d}$, and $f_j^{t_d}$ in Eq.(3)). K's value is determined by the elbow method [16]; we find that K=16 effectively separates the files.

Sub-step 3.2: Parallel Optimizing. The Push Task Scheduler distributes files evenly among P parallel optimization solvers after clustering. Each solver can push the same total number of replicas, which is an even share of the maximum number of replicas to be pushed in the lookup table. For instance, if there are five solvers and a maximum budget of 100 replicas, each solver can allocate 20. This parallel computation reduces the computational complexity to $O(M/P)$. Finally, each solver determines the number of replicas for each file and sends the results to the edge allocator.

Sub-step 3.3: Edge Allocation. The edge allocator is responsible for deciding which edge node each replica is pushed to. It maintains a table representing the edge node priorities in a push period for all files, where each row is an individual node. To build this table, each edge node periodically reports its content requests and their available cache space to the edge allocator [6], [8]. The priority order of edge nodes in the table is then obtained according to the sequence of requests received by the data center from edge nodes, *i.e.*, the first entry in the table is the node that sent the first request during the push period. Importantly, for scalability, the node priority is global, and *not* specific to each file. In practice, this results in content being pushed to the most popular edge nodes, as these tend to most frequently issue requests to the data center. The table is periodically refreshed; after the replicas have been pushed to an edge node, the edge allocator removes it from the table.

Next, we explain how the edge allocator finally selects which nodes to push each replica to. Recall, the optimization solvers output the number of replicas to push for each file, and the file ranking results list the predicted popularity of each file. Subsequently, the edge allocator iterates over the file ranking list and, for each file, selects the corresponding required number of nodes from the prioritized edge node list. For instance, if the number of replicas for a file is 10 (taken from Step 2 above), the edge allocator selects the top 10 prioritized edge nodes. Fig. 8 illustrates the edge allocator that chooses the target nodes for two files. Initially, it selects the top eight nodes for File 2 (red dashed box), then selects seven nodes for File 3 based on node rank (green dashed box).

IV. Evaluation

We deployed SDPush on a top user-generated video platform with millions of daily active users, evaluating its efficacy through trace-driven emulations and our real deployment.

A. Evaluation Methodology

1) Parameter Settings: For the popularity prediction model, we set the dimension of the embedding vector of each feature to 8. The deep component consists of two fully connected layers (each with a size of 128). The compressed interaction network consists of 3 layers, and each layer outputs the feature map with a size of 12. Then the outputs from the linear part, the compressed interaction network, and the deep component are concatenated. For the prediction model, the concatenated values are passed into a dense layer with 128 neurons and a linear function sequentially. The concatenated values are passed into a linear function for the ranking output. The output of the predicted and reference files is subtracted as the final output. All the feature layers use ReLu, except the output layer, which uses the linear function. For the Push Task Scheduler, if not specified, we set the number of optimization solvers to 100.

2) Baselines: We compare SDPush with the following baseline methods: (*i*) **Size Based (SB):** the number of pushed replicas is only related to the file size, which can be represented as $k_b s_j$ + q_b for file j. k_b and q_b are constants and we find the best (k_b, q_b) combination through a grid search. This is the previous method used on our deployed video platform. (*ii*) **Requests Based (RB) [6], [8]:** the number of pushed replicas is only related to the historical request count, which can be represented as $k_r r_j^p$ + q_r for file j. k_r and q_r are constants and we find the best (k_r, q_r) combination through the grid search method. (*iii*) **SDPush without the number of replicas limitation Lookup Table (SDPush-LT):** The upper bound of the total number of replicas pushed during each period remains the same (rather than vanilla SDPush, which adapts the limit per time period). The total number of replicas pushed per push period is $\frac{\Gamma}{N_p}$, where N_p is the number of push periods in a day. Γ is the total number of pushed replicas per day, equal to the vanilla SDPush. (*iv*) **SDPush without Popularity Predictor (SDPush-PP):** This is SDPush without the Popularity Predictor. Instead, it uses the number of requests for each file in the past hour as a prediction of its peak-period popularity. (*v*) **SDPush without Push Task Scheduler (SDPush-PTS):** This is SDPush without the Push Task Scheduler. Instead, the decision of the number of push replicas for each file is based on the Size Based method.

3) Evaluation Metrics: We evaluate SDPush using several metrics. (*i*) **Peak-period's Covered Requests:** This metric counts the sum of all requests served by edge nodes. This indicates the accuracy of the popularity prediction. The value of this metric is scaled to protect the privacy of the commercial system. (*ii*) **Data Center's 95th Percentile Traffic:** We collect all daily traffic samples and record the 95th percentile from them. 95th percentile pricing is widely used by CDNs [17]. Thus, reducing the 95th percentile traffic of the data center indicates reducing the bandwidth cost of CDN. We scale the value of data center traffic to protect the privacy of the commercial system. (*iii*) **Zero Request Replica Ratio**: The daily ratio of the number of deployed replicas without requests during the peak period vs. the number of all replicas at the edge. This metric captures the number of wasted replicas. (*iv*) **Magnification Ratio:** This metric is the ratio of the traffic generated by edge nodes each day vs. to the traffic required to push replicas to the edge nodes. This metric measures the cost-effectiveness of the pushing mechanism.

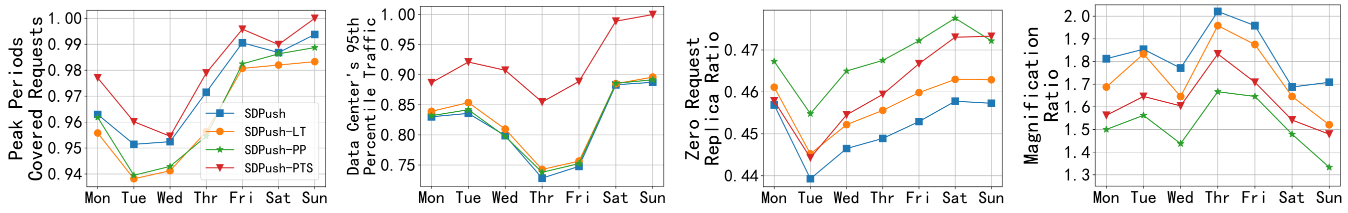
B. Trace-driven Simulations

1) Experimental Implementation: We randomly sample 20% of the video files that appear from December 6th, 2021, to December 19th, 2021, and record all requests for these files. Each file is encoded using specific encoding parameters (*i.e.*, resolution, codec, and quality), with a length of 1 to 4 minutes. This test dataset contains all the features mentioned in Section II-A. Consistent with the production environment, SDPush only pushes replicas from 00:00 to 20:00 in our trace-driven experiments. This is because the data center may reach its peak in the number of requests after 20:00. Thus, pushing files after 20:00 may run the risk of incurring additional data center pricing traffic. The push period is 10 minutes, the same as the settings in the production environment. The edge nodes periodically send file requests to the data center. Finally, The edge nodes adopt the Least Recently Used algorithm [18] as the cache eviction strategy to evict cached replicas.

2) Results: Fig. 9 shows the benefits of each method to the proactive push mechanism. We highlight three key takeaways.

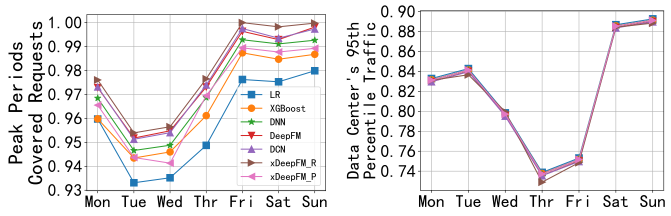
First, the Push Task Scheduler significantly decreases the data center's 95th percentile traffic with a slight reduction in peak-period's covered requests number (compared the blue line with the red line in Fig. 9(a) and Fig. 9(b)). This equates to a significant cost saving, due to the common use of 95th percentile pricing. The Popularity Predictor and the upper bound lookup table compensate for the reduced covered requests number. Consequently, SDPush achieves a reduction of 10.6% data center traffic at 95th percentile without reducing the number of covered requests.

Second, the Popularity Predictor and lookup table for the upper bound of the push replica number improve the peak-period's covered requests, and decrease the data center's 95th percentile traffic. Without the upper bound lookup table



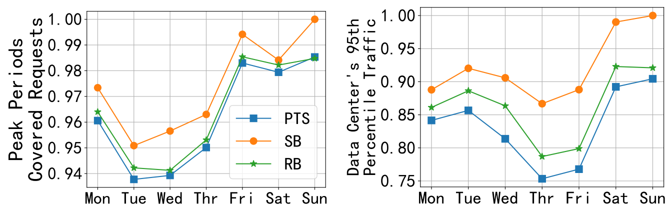
(a) Peak-Period's Covered Requests (b) 95th Percentile Traffic (c) Zero Request Copy Ratio (d) Magnification Ratio

Fig. 9: SDPush Overall Performance



(a) Peak-Period's Covered Requests (b) 95th Percentile Traffic

Fig. 10: Popularity Prediction Evaluation



(a) Peak-Period's Covered Requests (b) 95th Percentile Traffic

Fig. 11: Push Task Scheduling Evaluation

(orange line) and Popularity Predictor (green line), the peak-period's covered requests number is 1.1% and 0.8% lower than that of SDPush, respectively (Fig. 9(a)). The data center's 95th percentile traffic is 1.03% and 1.09% higher than SDPush, respectively (Fig. 9(b)).

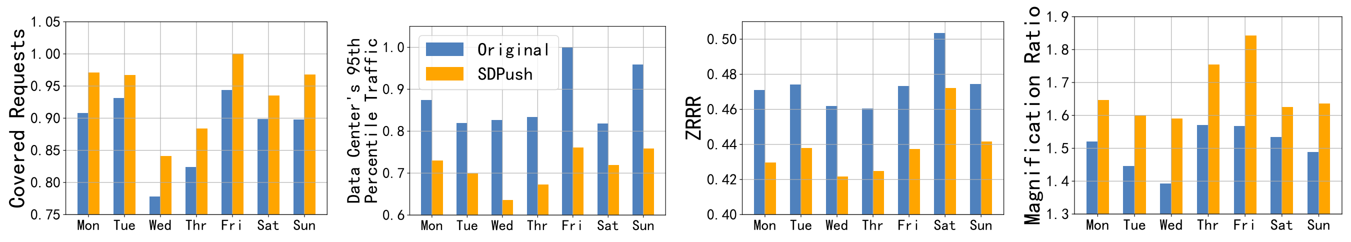
Third, Fig. 9(c) shows the zero request copy ratio. We see that all methods reduce the zero request replica ratio, and the Popularity Predictor (green line) plays the most important role. The Popularity Predictor and upper bound lookup table reduce the zero request replica ratio by filtering out files that are not worth pushing. On the other hand, since the Push Task Scheduler deploys more replications for high-benefit files, it reduces the deployment of low-benefit files, which also reduces the zero request replica ratio. Finally, Fig. 9(d) presents the magnification ratio. We see that all methods effectively increase the magnification ratio implying the benefit of replicating the files significantly exceeds to initial cost of pushing them.

3) Ablation Study: We first highlight the differing performance of the various **Popularity Prediction** models we experimented with. Fig. 10 shows the peak period's covered requests and the data center's 95th percentile traffic of all popularity prediction methods. We only vary the popularity prediction model — the other optimization methods remain consistent with SDPush. xDeepFM_R (used by SDPush) and xDeepFM_P indicate using the ranking and predicting models' results to rank files, respectively. The ranking results are finally fed to the edge allocator (shown in Fig. 8).

Compared with other methods, the xDeepFM-based feature extraction model increases the peak period's covered requests by 0.2%$\sim$2.2% (shown in Fig. 10(a)) , while saving the data center's 95th percentile traffic by 0.15%$\sim$0.86% (shown in Fig. 10(b)). On the other hand, the use of xDeepFM_R increases the covered requests of the peak period by 0.9% compared to xDeepFM_P. It also reduces the data center's 95th percentile traffic by 0.51%. This proves that ranking objectives can filter out unpopular files from the files at the junction of popular and unpopular than other methods.

Next, we inspect the **Push Task Scheduler** component. The Push Task Scheduler decides the appropriate number of replicas per file by fusing the popularity and size information of the files. We first verify how much performance improvement the Push Task Scheduler has compared to methods that simply use raw popularity (Requests Based) or size (Size Based). We find that the Push Task Scheduler results in a slight decrease in the peak period's covered requests (shown in Fig. 11(a)), but a significant decrease in the data center's 95th percentile traffic (shown in Fig. 11(b)). This represents a notable cost saving, due to 95th percentile bandwidth pricing. The Push Task Scheduler decreases the data center's 95th percentile traffic by 9.73% and 3.49%, with 1.28% and 0.27% loss of the covered requests compared with the Size Based and Request Based methods, respectively. The decrease in the data center's 95th percentile traffic implies the Push Task Scheduler effectively combines the advantages of Request Based and Size Based methods, and finds a more beneficial decision for the replica number per file.

Another optimization of the Push Task Scheduler is to reduce the computational complexity using file clustering and parallel optimization solvers. We find that decreasing the solver number P (*i.e.*, increasing in the number of files in each solver) results in a linear increase in computation time, but a slight decrease in the data center's 95th percentile traffic. When we decrease the solver number from 400 to 50, the computing time decreases from 2.89 seconds to 0.36 seconds, while the data center's 95th percentile traffic only increases by 0.27%. Thus, in our evaluation setup, we set P to 100, which is mentioned in Section IV-A1. We believe this a suitable trade-off, considering the benefits of scalability.



(a) Peak-Period's Covered Requests (b) 95th Percentile Traffic (c) Zero Request Replica Ratio (d) Magnification Ratio

Fig. 12: Real-World Deployment Performance

C. Real-world Deployment

1) Experimental Implementation: We have deployed SDPush on the production network of Xigua and run between the 22nd and 28th August 2022. To measure its effectiveness, we randomly select 25% files and use SDPush to push them to the edge nodes. The remaining files use the original strategy (Size Based).

2) Results: Fig. 12 shows the comparison between the original Size Based mechanism and SDPush. We again plot all four evaluation metrics. The results show that SDPush saves 12.1%$\sim$23.9% of 95th percentile traffic, and increases the covered requests at edge nodes by 4.07%$\sim$8.09%. This reduces subsequent bandwidth cost, which is typically priced at the 95th percentile. Fig. 12(c) shows the zero request replica ratio is reduced by 6.23%$\sim$8.77%, and the magnification ratio increases by 8.34%$\sim$17.59%. The results show SDPush effectively improves the probability that edge nodes serve clients, and reduces the data center's bandwidth cost.

V. Related Works

Edge-assisted video delivery for quality increase and bandwidth saving: Research has tried to improve video content delivery by using diverse smart solutions deployed at the edge, including adaptivity [19] [20] and edge caching [21] [22]. Researchers have deployed cache admission mechanisms at the edge, whereby they only cache videos with high revenue, avoiding the problem of frequent cache replacement. Such works can be divided into three groups: (*i*) frequency-based [23], [24]; (*ii*) size-based [25], [26]; and (*iii*) video-based [4]. However, these works only perform cache admission based on the information perceived by a single edge node. To further improve caching efficiency, other works take into account decisions by multiple edge caches. Thus, several centralized [27]–[29] and decentralized [30]–[33] cooperative edge caching schemes have been proposed. In these schemes, the edge nodes decide whether to cache a video according to other edge nodes' information. However, it is difficult for existing solutions to coordinate video information (*i.e.*, popularity and size), leading to poor decisions. Additionally, some works design cache eviction algorithms for edge nodes to reduce the traffic from the data center [34]–[38]. These predict the duration of file popularity and replace files with a rapid popularity decay. In contrast to the above reactive caching strategies, many works focus on proactive caching methods [6], [8], [9]. These solutions relieve the pressure on the data center during the peak period by proactively pushing popular videos to edge nodes during off-peak periods. However, these schemes are yet to incorporate the popularity and size heterogeneity among files. Thus, edge networks have lower cache utilization due to replicating the wrong objects.
Popularity prediction by fusion of multi-dimensional features: File popularity is related to multiple features. Several works have proposed techniques to integrate multi-dimensional features to predict content popularity [4], [39], [40]. However, such methods fail to leverage combined features for popularity prediction effectively. Nowadays, many deep learning-based models, such as DCN [14], DeepFM [13], and xDeepFM [12], have excellent performance for recommendation systems. Their feature extraction methods can extract high-order cross-features of content to improve the models' performance in predicting clients' click-through rates. However, all the above works aim to predict the precise popularity of files. The proactive push mechanism also needs to predict the relative popularity between multiple files to determine which files should be pushed to the edge network more urgently under constraints.

VI. Conclusions

In conclusion, this paper introduced SDPush, a novel smart data-driven proactive push mechanism, devised to mitigate peak-period data center traffic usage to the lightweight edge nodes. By strategically pushing files with high peak-period traffic to edge nodes during non-peak periods, SDPush effectively alleviates the resource scarcity problem of lightweight edge nodes. Unlike conventional proactive caching methods, SDPush takes into account both the popularity of each file and the estimated benefits of pushing them, offering a holistic approach to file selection. By evaluating the proposed solution using real-world traces and operational deployments, we have shown that SDPush outperforms existing proactive push mechanisms in terms of saving data center traffic consumption, thereby reducing CDN costs.

Acknowledgments

We express our appreciation to Professor Jiang Yong and the engineers at ByteDance for their constructive assistance in completing this paper. This work is supported by the Major Key Project of PCL under grant No. PCL2023A06. Prof. Muntean thanks Science Foundation Ireland for grants 12/RC/2289_P2 (Insight) and 21/FFP-P/10244 (FRADIS).

References

[1] A. Clouder. (2018) Content delivery acceleration and cost reduction with p2p cdn (pcdn). [Online]. Available: https://www.alibabacloud.com

[2] Lumen. (2023) Dynamically adjust to changes in consumer demand. [Online]. Available: https://www.lumen.com/en-us/industries/media-entertainment.html

[3] PacketFabric. Packetfabric cloud router: The industry's most scalable and performant multi-cloud connectivity. [Online]. Available: https://packetfabric.com/cloud-router

[4] Y. Guan, X. Zhang, and Z. Guo, "CACA: learning-based content-aware cache admission for video content in edge caching," in *Proceedings of the 27th ACM International Conference on Multimedia (ACM MM 19)*. ACM, 2019, pp. 456–464.

[5] Cloudstream. (2023) Pcdn. [Online]. Available: https://www.cloudstream-tech.com/en/pcdn-2/

[6] Y. Zhang, C. Gao, Y. Guo, K. Bian, X. Jin, Z. Yang, L. Song, J. Cheng, H. Tuo, and X. Li, "Proactive video push for optimizing bandwidth consumption in hybrid CDN-P2P VoD systems," in *2018 IEEE Conference on Computer Communications (INFOCOM 18)*, 2018, pp. 2555–2563.

[7] R.-X. Zhang, M. Ma, T. Huang, H. Pang, X. Yao, C. Wu, J. Liu, and L. Sun, "Livesmart: A qos-guaranteed cost-minimum framework of viewer scheduling for crowdsourced live streaming," in *Proceedings of the 27th ACM International Conference on Multimedia (ACM MM 19)*, 2019, pp. 420–428.

[8] Y. Guo, Y. Zhang, Z. Yang, K. Bian, H. Tuo, and Y. Dai, "Atdps: An adaptive time-dependent push strategy in hybrid cdn-p2p vod system," in *2018 IEEE International Conference on Communications (ICC 18)*. IEEE, 2018, pp. 1–6.

[9] Y. Zhang, K. Bian, H. Tuo, B. Cui, L. Song, and X. Li, "Geo-edge: Geographical resource allocation on edge caches for video-on-demand streaming," in *2018 4th International Conference on Big Data Computing and Communications (BIGCOM 18)*. IEEE, 2018, pp. 189–194.

[10] R. Zhan, C. Pei, Q. Su, J. Wen, X. Wang, G. Mu, D. Zheng, P. Jiang, and K. Gai, "Deconfounding duration bias in watch-time prediction for video recommendation," in *The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022 (KDD 22)*, 2022, pp. 4472–4481.

[11] J. Deng, F. Cuadrado, G. Tyson, and S. Uhlig, "Behind the game: Exploring the twitch streaming platform," in *2015 International Workshop on Network and Systems Support for Games (NetGames 15)*. IEEE, 2015, pp. 1–6.

[12] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xdeepfm: Combining explicit and implicit feature interactions for recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 18)*, 2018, pp. 1754–1763.

[13] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv preprint arXiv:1703.04247*, 2017.

[14] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 17)*, 2017, pp. 1–7.

[15] N. R. Draper and H. Smith, *Applied regression analysis*. John Wiley & Sons, 1998, vol. 326.

[16] M. Syakur, B. Khotimah, E. Rochman, and B. D. Satoto, "Integration k-means clustering method and elbow method for identification of the best customer profile cluster," in *IOP Conference Series: Materials Science and Engineering*, vol. 336, no. 1. IOP Publishing, 2018, p. 012017.

[17] R. Stanojevic, N. Laoutaris, and P. Rodriguez, "On economic heavy hitters: Shapley value analysis of 95th-percentile pricing," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 75–80.

[18] H. Johnson and J. Larson, "Data management for microcomputers," in *1979 Compcon Fall*. IEEE Computer Society, 1979, pp. 191–192.

[19] X. Ma, Q. Li, Y. Jiang, G.-M. Muntean, and L. Zou, "Learning-based joint qoe optimization for adaptive video streaming based on smart edge," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1789–1806, 2022.

[20] X. Ma, Q. Li, L. Zou, J. Peng, J. Zhou, J. Chai, Y. Jiang, and G.-M. Muntean, "Qava: Qoe-aware adaptive video bitrate aggregation for http live streaming based on smart edge computing," *IEEE Transactions on Broadcasting*, vol. 68, no. 3, pp. 661–676, 2022.

[21] H. S. Goian, O. Y. Al-Jarrah, S. Muhaidat, Y. Al-Hammadi, P. D. 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.

[22] A. Zhang, Q. Li, Y. Chen, X. Ma, L. Zou, Y. Jiang, Z. Xu, and G.-M. Muntean, "Video super-resolution and caching—an edge-assisted adaptive video streaming solution," *IEEE Transactions on Broadcasting*, vol. 67, no. 4, pp. 799–812, 2021.

[23] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 52–66, 2015.

[24] G. Einziger, R. Friedman, and B. Manes, "Tinylfu: A highly efficient cache admission policy," *ACM Transactions on Storage (ToS)*, vol. 13, no. 4, pp. 1–31, 2017.

[25] D. Starobinski and D. Tse, "Probabilistic methods for web caching," *Performance evaluation*, vol. 46, no. 2-3, pp. 125–137, 2001.

[26] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "*AdaptSize*: Orchestrating the hot object memory cache in a content delivery network," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 483–498.

[27] A. Larbi, L. Bouallouche-Medjkoune, and D. Aissani, "Improving cache effectiveness based on cooperative cache management in manets," *Wireless Personal Communications*, vol. 98, no. 3, pp. 2497–2519, 2018.

[28] E. E. Ugwuanyi, S. Ghosh, M. Iqbal, T. Dagiuklas, S. Mumtaz, and A. Al-Dulaimi, "Co-operative and hybrid replacement caching for multi-access mobile edge computing," in *2019 European Conference on Networks and Communications (EuCNC 19)*. IEEE, 2019, pp. 394–399.

[29] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS 18)*. IEEE, 2018, pp. 1–6.

[30] W. Jiang, G. Feng, S. Qin, T. S. P. Yum, and G. Cao, "Multi-agent reinforcement learning for efficient content caching in mobile d2d networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1610–1622, 2019.

[31] 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 *2020 IEEE Conference on Computer Communications (INFOCOM 20)*. IEEE, 2020, pp. 2499–2508.

[32] L. Wang, G. Tyson, J. Kangasharju, and J. Crowcroft, "Milking the cache cow with fairness in mind," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2686–2700, 2017.

[33] J. Peng, Q. Li, X. Ma, Y. Jiang, Y. Dong, C. Hu, and M. Chen, "Magnet: Cooperative edge caching by automatic content congregating," in *Proceedings of the ACM Web Conference 2022 (TheWebConf 22)*, 2022, pp. 3280–3288.

[34] G. Vietri, L. V. Rodriguez, W. A. Martinez, S. Lyons, J. Liu, R. Rangaswami, M. Zhao, and G. Narasimhan, "Driving cache replacement with *ML-based LeCaR*," in *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.

[35] L. V. Rodriguez, F. Yusuf, S. Lyons, E. Paz, R. Rangaswami, J. Liu, M. Zhao, and G. Narasimhan, "Learning cache replacement with cacheus," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 341–354.

[36] T. Zong, C. Li, Y. Lei, G. Li, H. Cao, and Y. Liu, "Cocktail edge caching: Ride dynamic trends of content popularity with ensemble learning," in *2021 IEEE Conference on Computer Communications (INFOCOM 21)*. IEEE, 2021, pp. 1–10.

[37] N. Megiddo and D. S. Modha, "*ARC*: A *Self-Tuning*, low overhead replacement cache," in *2nd USENIX Conference on File and Storage Technologies (FAST'03)*, 2003, pp. 52–66.

[38] V. Kirilin, A. Sundarrajan, S. Gorinsky, and R. K. Sitaraman, "Rl-cache: Learning-based cache admission for content delivery," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2372–2385, 2020.

[39] C. Koch, G. Krupii, and D. Hausheer, "Proactive caching of music videos based on audio features, mood, and genre," in *Proc. of the 8th ACM on Multimedia Systems Conference (MMsys 17)*, 2017, pp. 100–111.

[40] L. Tang, Q. Huang, A. Puntambekar, Y. Vigfusson, W. Lloyd, and K. Li, "Popularity prediction of facebook videos for higher quality streaming," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 111–123.