Online Midgress-Sensitive Traffic Allocation for Percentile Charging in Pracitcal CDNs

Huiyou Zhan[†‡], Haisheng Tan[†‡], Huang Xu[§], Chi Zhang[†], Hongqiu Ni[†], Pengfei Zhang[†],
Weihua Shan[§], Xiang-Yang Li[†‡]
[†] CAS Key Lab of Wireless-Optical Communications, University of Science and Technology of China, China
[‡] USTC-Deqing Alpha Innovation Research Institute, Zhejiang, China
[§] Algorithm Innovation Lab, Huawei Cloud Computing Technologies Co., Ltd. Xi'an, China

2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS) | 979-8-3503-9973-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/IWQOS57198.2023.10188693

***Abstract*—The traffic bandwidth costs comprise a significant amount of operating expenditure in CDNs, induced by the traffic from the end-users to edge servers (edge cost) and from the edge to the center servers (midgress cost). Traffic allocation is the main approach to minimizing the total bandwidth cost. The joint optimization of the total costs is challenging, specifically when the percentile charging mechanism as well as some other practical issues are considered, such as the dynamicity of midgress traffic and the granularity of traffic allocation. In this work, based on our novel miss ratio prediction mechanism, we propose the first online framework, named `Iris`, jointly optimizing the edge and midgress costs under the 95th percentile charging in commercial CDNs. `Iris` can theoretically achieve a competitive ratio of $1 + \frac{p_e}{\beta \cdot p_c}$, when the miss ratio of all domains is set as β. Here p_e and p_c are the unit bandwidth price of the edge and midgress cost, respectively. `Iris` is tolerant to prediction errors which can be deployed in practical CDN systems. Extensive experiments based on real data indicate that `Iris` can dramatically reduce bandwidth costs by about 8.149% compared with the SOTA schemes, potentially saving millions of dollars per month for our large-scale commercial CDN collaborator.**

I. Introduction

Content Delivery Networks (CDNs), deploying a large number of edge cache servers worldwide, enable end-users to obtain desired content nearby, which significantly reduces network congestion and improves user experience. Nowadays, with the explosion of online applications, *e.g.*, high-definition videos, online music and gaming, CDNs carry massive network traffic globally and suffer an increasingly higher bandwidth cost [1]. It is reported that data transmission in CDNs will reach 250 EB recently, taking more than 72% of the network transmission [2]. As a consequence, a CDN system could incur a bandwidth cost of millions of dollars per month, and even a small fraction of bandwidth reduction translates into massive monetary savings.

CDN companies pay bandwidth costs to Internet Service Providers (ISP) for all outbound traffic, which consists of *edge traffic* and *midgress traffic*. When serving an end-user, a CDN allocates the request to one selected edge server. If the requested content is available at the edge, a cache hit occurs. Otherwise, it is a miss and the content will be fetched from the remote center server to the edge. The *edge traffic* at each time interval is the sum of the requested file sizes returned from the edge server to users, and the *midgress traffic* is the sum of the file sizes sent from the center server to edge servers. Both edge traffic and midgress traffic follow the industry-standard 95th percentile charging scheme to pay bandwidth fees. Specifically, the ISP records the traffic bandwidth of the server at all time intervals to form a traffic sequence and takes the 95th percentile of the sequence as the *charging bandwidth*. The 5% time intervals when the bandwidth is higher than the charging bandwidth are free-of-charge. The charging bandwidth multiplied by the unit price is the bandwidth cost of a server. This scheme can prevent excessive penalties on CDN companies due to occasional surges in traffic [3], and it has become the commercial charging scheme widely adopted.

979-8-3503-9973-8/23/$31.00 2023 IEEE

The traffic allocation strategy plays a critical role in reducing data transmission and bandwidth costs. Most existing studies only focus on the optimization of either the bandwidth cost of edge traffic or that of midgress traffic, which are called *edge cost* and *midgress cost*, respectively. Under the 95th percentile charging, the reduction of edge costs usually requires to use of the free-of-charge intervals of some edge servers by turns to reduce the allocated bandwidth of other edge servers in most intervals [1]. However, frequent changes in this allocation strategy increase miss ratios of requests on edge servers, resulting in high midgress costs. At the other extreme, the allocation strategy may make requests for the same content be satisfied by a fixed server to improve the hit ratio, thereby reducing the midgress cost [4]. This approach may significantly increase edge costs due to the inability to adapt to sudden traffic peaks. Therefore, both of the above intuitive strategies will incur high total costs, and joint optimization of the edge and midgress costs is desired. However, it is full of challenges under the 95th percentile charging scheme in the following three ways:

- **Non-linearity of charging scheme:** The percentile charging scheme introduces some free time intervals, and this non-linearity makes optimization extraordinarily complicated in large-scale CDNs.
- **Dynamicity of Midgress traffic:** When jointly optimizing edge costs and the midgress cost, it is difficult to evaluate the performance of real-time allocation policies due to the dynamicity of midgress traffic. As the caching data on edge servers changes over time, the midgress traffic that the center server send to edge servers due to cache miss may vibrate.

Moreover, our analysis on a commercial CDN shows that as the cache space occupied by different types of requests varies, their cache miss ratios change at different rates.

- **Limitations of allocation policies:** Traffic allocation policies are implemented through the Domain Name System (DNS), so a practical approach can not divide the traffic infinitely but with an allocation granularity. Moreover, due to the limitation of hardware, the system can not accurately follow our policy which will also influence its performance.

In this work, we tackle the above challenges by analyzing the opportunity of bandwidth cost saving on 95th percentile charging, designing traffic allocation policies that reduce total bandwidth costs by explicitly incorporating midgress cost and making full use of the free-of-charge intervals on edge servers. Our main contribution is as follows:

- **Practical model.** To the best of our knowledge, this is the first work to minimize both the edge and midgress costs under the percentile charging in CDNs. We take the practical issues into account, which are full of challenges due to the nonlinear charging as well as the complicated mutual influence between the edge and midgress costs.
- **Miss ratios prediction.** To estimate the midgress costs of traffic allocation policies, we design a mechanism that can accurately model and predict the cache miss ratio when multiple types of traffic share a single cache. Extensive experiments on commercial CDNs demonstrate that our mechanism has a prediction error of 1.804% on average.
- **Online midgress-sensitive joint optimization framework.** Based on the midgress cost prediction, we devise the first online framework `Iris` that can jointly optimize the edge and midgress costs. We prove that when the miss ratio is a constant β, the competitive ratio of `Iris` is $1+\frac{p_e}{\beta \cdot p_c}$, where p_e and p_c are the bandwidth unit price of edge servers and center server, respectively. We further adapt `Iris` to make it tolerant to practical issues such as prediction errors.
- **Evaluation based on commercial CDN.** Extensive experiments on a large-scale commercial CDN trace illustrate that `Iris` can reduce bandwidth costs by about 8.149% compared to the state-of-the algorithms, potentially saving millions of dollars per month. Furthermore, `Iris` performs consistently well on various settings of the granularity, the number as well as the cache sizes of servers.

II. System Model and Problem Formulation

Cache System. Consider a CDN system with K edge cache servers and one center server c. For each edge server k, the bandwidth capacity is B_k and the cache size is C_k. We assume that the bandwidth capacity and cache size of the center server are unlimited. Each cache server adopts a replacement strategy, *e.g.*, one of the extensions or variants of LRU [5].

Traffic Allocation. The traffic allocation process in CDNs is shown in Fig. 1. Users access network resources with domain names. A CDN system sends the requested domain name to the DNS server, which stores an IP table mapping domain names to edge servers' IP addresses. Based on the traffic allocation framework, the DNS server adjusts the proportion of the traffic for one domain to several edge servers by controlling the number of IP addresses (Fig. 1④) [1]. As shown in Fig. 1.①, both edge server 1 and 2 receive half of the Domain 1's traffic. The user accesses one of the IP addresses to obtain the resources (Fig. 1.②). If the edge server has the requested content, it delivers the content to the user directly. If not, a cache miss occurs and the edge server fetches the content from the center server, where we assume all contents are stored (Fig. 1.③).

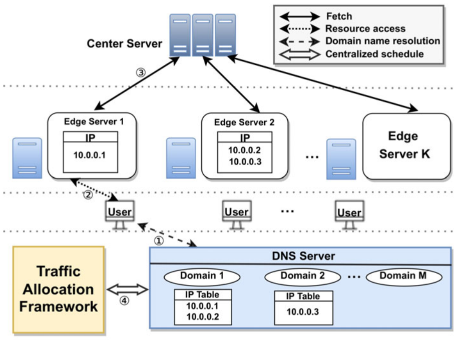


Fig. 1. An illustration of the traffic allocation process in CDNs.

Percentile Charging Mechanism. The bandwidth cost on a server can be calculated as $p \times x$, where x is the charging bandwidth for a charging period and p is the unit bandwidth price. In practical CDNs, the bandwidth unit price p_c of a center server is about two times that of edge servers p_e due to higher construction cost. Vector $V = \langle v_1, v_2, \cdots, v_t, \cdots, v_T \rangle$ denotes the total traffic of all T time intervals, and $V_k = \langle v_1^k, \cdots, v_T^k \rangle$ denotes the traffic allocated to server k. Note that in our online setting, each v_t is unknown until interval t. Under the percentile charging, after obtaining the traffic sequence V_k of server k, we sort V_k in ascending order and let the $\lfloor q \times T \rfloor$-th value be the charging bandwidth represented as $Q(V_K, q)$. Here q is the charging quantile.

Problem Formulation. At interval t, we have h_t^d traffic for domain d. Set $\beta_t^{k,d}$ as the miss ratio of edge server k. The bandwidth allocated to edge server k at t is $v_t^k = \sum_d n_t^{k,d} g h_t^d$, where g is the allocation granularity and $n_t^{k,d}$ is an integer ($n_t^{k,d} \in [0, 1/g]$). Our goal is to minimize the total costs by determining how many requests are allocated to each edge server at each interval. The problem can be formalized as:

$$\text{minimize } p_e \sum_{k=1}^{K} Q(V_k, 0.95) + p_c Q(V_c, 0.95)$$

$$s.t. \quad v_t^k = \sum_d n_t^{k,d} g h_t^d \le B_k, \qquad \forall k, t \quad (1)$$

$$v_t = \sum_k v_t^k = \sum_d h_t^d, \qquad \forall t \quad (2)$$

$$v_t^c = \sum_k \sum_d \beta_t^{k,d} n_t^{k,d} g h_t^d, \qquad \forall t \quad (3)$$

$$n_t^{k,d} \in \{0, 1, 2, \cdots, \}, \qquad \forall d, k, t \quad (4)$$

[1] Therefore, traffic bandwidth for a domain can only be allocated at a certain granularity g, typically $g \in [1\%, 5\%]$.

When the traffic arrives at interval t, we give a real-time traffic allocation scheme, where the allocated bandwidth to each edge server can not exceed its capacity B_k (Constraint (1)). Traffic for each interval needs to be all allocated (Constraint (2)). The traffic bandwidth v_t^k on the edge server k at interval t should be the sum of an integer multiple of the domain traffic granularity (Constraint (1)(4)). And the center server's bandwidth v_t^c is the sum of the midgress traffic of all domains (Constraint (3)).

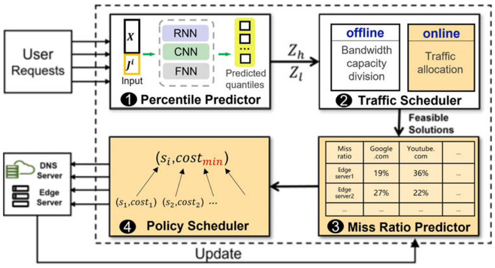
Problem Hardness. Even in an offline setting with perfect knowledge of traffic, the traffic allocation problem is still hard.

Theorem 1. *There is no polynomial-time algorithm that can achieve an optimal solution for the traffic allocation problem based on the 95th charging scheme when the miss ratio is constant $\beta \in (0, 1]$, unless P=NP.*

Proof. Please refer to Appendix A. □

III. Design of Our Framework Iris

The key idea of Iris is to solve the joint optimization problem in layers. As shown in Fig. 2, the first layer contains a scheduler and a percentile predictor to guide traffic allocation to each edge server at each time interval. The second layer includes a cache miss ratio predictor and a selector to select the estimated best allocation policy. Totally Iris consists of 4 key components. **Percentile Predictor** quantifies opportunities for bandwidth cost savings on edge servers, which are related to the 95th percentile value of the total traffic sequences and the theoretical optimal percentile value. We present a CRNN-based model to predict these two parameters accurately (in Sec. III-A). **Traffic Scheduler** determines the traffic allocated to each edge server at each interval. Based on the two predicted key parameters, we design a region-based traffic allocation algorithm and further prove that the competitive ratio of this algorithm is $1 + p_e/(\beta \cdot p_c)$, where β is the missing ratio of all domains on the edge server (in Sec. III-B). **Miss Ratio Predictor** estimates the midgress cost of an allocation policy. It takes the output of the first-layer framework as soft constraints to generate a set of feasible allocation policies. We design a traffic descriptor that can estimate the midgress cost of these feasible policies based on historical statistics in real-time (in Sec. III-C). **Policy Selector** selects the policy with the lowest estimated cost to configure the DNS server, which is highly tolerant to the miss ratio prediction errors. Next, we will present a detailed description of each component.



Miss ratio	Google .com	Youtube. com	...
Edge server1	19%	36%	...
Edge server2	27%	22%	...
...	...	...	...

Fig. 2. The design of our traffic allocation framework Iris.

A. Percentile Predictor

Due to network traffic dynamics, it is hard to estimate the percentiles accurately in advance, especially in the early days of a charging period. We have a key observation that the scale of the traffic typically has an increasing or decreasing trend depending on the historical cycle and the periodical commercial market. There is also a day-to-day variance within a charging period, which can be quantitatively identified by comparing the percentiles of two adjacent days. We propose a CRNN-based prediction model fully utilizing the above characteristic. We here integrate two main modifications: 1) we predict a list of quantiles $\vec{q} = \{q_1, q_2, \cdots q_i, \cdots\}$, where $q_i \in \{0.60, 0.61, ..., 0.98, 0.99\}$. Therefore, the predicted traffic value is a real-valued vector $\vec{\rho}$ instead of a scalar ρ. Each element of $\vec{\rho}$ is a traffic value that corresponds to a quantile in $\vec{q}$, *e.g.*, ρ_q is the predicted traffic value of the q quantile. The loss function is $\mathbf{L} = ||\vec{\rho} - \vec{y}^q||^2$, where $\vec{y}^q$ is a vector of ground truth traffic values. 2) We concatenate the trend of the traffic as features into the input matrix X of our model. Specifically, a vector J^i is calculated as the percentile traffic values of i-th row of traffic sequence of X, where each element J_q^i of J^i is a predicted percentile traffic value with $q \in \vec{q}$. The vector J^i is concatenated into the i-th row of the input matrix X.

The performance of our model is evaluated by the average prediction precision, defined as $p = 1 - \frac{1}{K} \sum_{i=1}^{K} \frac{\sum_j |\rho_j^i - y_j^i|}{\sum_j y_j^i}$, where ρ_j^i is the predicted j percentile of group i, y_j^i is the corresponding ground truth, and K is the number of edge server groups. We compare the performance of our model with the original CRNN model [6], RNN model [7] and Percentile model [8]. As shown in Fig. 3, our model predicts the percentiles accurately even in the early stage of a month, and outperforms other models.

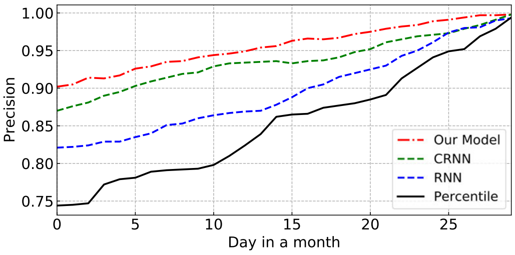


Fig. 3. The precision of the Quantile Prediction Model.

We use this model to predict the two key parameters of total traffic sequence V in a charging period, which are respectively 95th percentile and $\max(100 - K \times 5, 0)$-th percentile value, which has been proved to be the optimal charging bandwidth under the 95th charging mechanism without considering bandwidth capacity limitation [9].

B. Traffic Scheduler

Traffic scheduler determines how much traffic should be allocated to each edge server at each interval to reduce the overall edge cost. The design of traffic scheduler is based on the critical insight: 1) Neither the edge cost nor the midgress cost is directly related to the free-of-charge intervals. 2) The traffic allocation policy will not increase the edge costs, as long as the traffic allocated to each edge server does not exceed its own charging bandwidth.

Accordingly, the traffic scheduler will calculate an estimated charging bandwidth for each edge server, named *red line*. Naturally, the number of intervals where the allocated traffic exceeds the red line should be no more than $5\% \times T$. However, there may be a deviation in the prediction of the percentile value in practice. Thus, we should use the bandwidth capacity below the red line conservatively to avoid the unexpected increase in the charging bandwidth. Next, we will introduce two key strategies of the traffic scheduler for this purpose: 1) bandwidth capacity division based on the percentile predictor and 2) traffic allocation in real time.

Bandwidth Capacity Division. We divide the bandwidth capacity of edge server k into three zones: red, blue, and green, separated by the red line (Rl_k) and the blue line (Bl_k). Due to this key procedure, we vividly name our traffic allocation framework as `Iris`. The red zone above the red line indicates the capacity with the number of usages less than $5\% \times T$. Below the blue line is the blue zone, which represents the capacity that can be used at will. Between the red and blue lines is the green zone, which is further divided equally into multiple gears. The two key parameters predicted by the percentile predictor are represented by Z_h and Z_l, which are respectively the sum of the red and blue lines of edge servers. In Algorithm 1, we make the red zone of each edge server as numerically equal as possible, which helps to balance the number of red zones used in the edge server to cope with the possible burst traffic peak (Line 1-5). Accordingly, we divide the total blue line into blue zones according to the ratio of the red line of an edge server to the total red line (Line 6-7). And then, we divide the green zone evenly into N gears, which is an adjustable hyperparameter (Line 9-11).

```
Algorithm 1: Bandwidth Capacity Division
   Input: $\{B_k\}$, $Z_h$, $Z_l$, number of the gears: $N$
1  Sort $\{B_k\}$ in descending order
2  $B \leftarrow \sum_k B_k$, $RedSum \leftarrow B - Z_h$
3  for $k = K$ to $1$ do
4      $Rl_k \leftarrow B_k - \min(RedSum/k, B_k)$
5      $RedSum \leftarrow RedSum - \min(RedSum/k, B_k)$
6  for $k \in [1, K]$ do
7      $Bl_k \leftarrow Z_l \cdot \frac{Rl_k}{\sum_k Rl_k}$, $Gear_k[0].id \leftarrow k$
8      $Gear_k[0].size \leftarrow Bl_k$, $Gear_k[0].value \leftarrow Bl_k$
9      for $n \in [1, N]$ do
10         $Gear_k[n].size \leftarrow \frac{Rl_k - Bl_k}{N}$, $Gear_k[n].id \leftarrow k$
11         $Gear_k[n].value \leftarrow Bl_k + n \cdot \frac{Rl_k - Bl_k}{N}$
12 return $\{Rl_k\}, \{Bl_k\}, \{Gear_k\}$
```

Online Traffic Allocation. The key idea of the online traffic allocation algorithm is to use the gear with the lowest value and the least usage number for intervals with traffic that may affect the charging bandwidth, and to fill up the bandwidth capacity of some servers for the 5% intervals with the largest traffic. In particular, Algorithm 2 first calculates the total traffic v_t of the current interval, the sum of the red lines of edge servers as $Z_h^{'}$ and the sum of the blue lines as $Z_l^{'}$ (Line 3-4). Then, according to the numerical relationship between v_t and $Z_h^{'}$ and $Z_l^{'}$, the allocation strategy is considered in three cases: 1) When $v_t > Z_h^{'}$, fill the blue zones of each server first, and allocate the remaining traffic as much as possible to the server with the least number of times of use at the highest gear (Line 5-11). 2) When $Z_l^{'} < v_t < Z_h^{'}$, we use the lowest gear of each server first, and give priority to the gear of the server that is used less frequently. If the traffic is still not fully allocated, higher gears of edge servers are used (Line 12-21). 3) When $v_t < Z_l^{'}$, we use a load balancing strategy, which allocates the total traffic according to the ratio $Bl_k / Z_l^{'}$ (Line 22-25). After the traffic allocation is completed, the statistics of the gear are updated. If the usage times of a certain gear exceed $F = 5\% \times T$, the blue line of the edge server is modified to the bandwidth value of this gear (Line 26-28).

Analysis. Let $\mathcal{A}(\mathcal{P})$ and $\text{OPT}(\mathcal{P})$ be respectively the total cost of our online algorithm $\mathcal{A}$ and offline optimal solution of Problem $\mathcal{P}$. If the percentile predictor is accurate for Z_h, then we have the following theorem.

Theorem 2. $\mathcal{A}(\mathcal{P}) \leq (1 + \frac{p_e}{\beta p_c})$ *OPT*$(\mathcal{P})$ *when the miss ratio of user requests for each domain name on edge servers is a constant* β.

Proof. Since $\beta_t^{k,d} = \beta$, the midgress traffic of time interval i is $v_t^c = \sum_k \sum_d (\beta \cdot n_t^{k,d} gh_t^d) = \beta \sum_k v_t^k = \beta v_t$. Therefore, the midgress cost of the solution of algorithm $\mathcal{A}$ and the optimal solution under the 95 charging mechanism are both $p_c Q(V_c, 0.95) = p_c \beta Q(V, 0.95) = p_c \beta Z_h$. Thus $\text{OPT}(\mathcal{P}) \geq p_c \beta Z_h$. For algorithm $\mathcal{A}$, we only use the red zone of edge servers when the traffic is greater than Z_h. When the prediction of Z_h is accurate, only $5\% * T$ intervals have traffic greater than Z_h, which means that the charging bandwidth of each edge server is not greater than Rl_k. Therefore, the bandwidth cost of edge servers satisfies $p_e \sum_k Q(V_k, 0.95) \leq p_e \sum_k Rl_k = p_e Z_h$. Thus, $\frac{\mathcal{A}(\mathcal{P})}{\text{OPT}(\mathcal{P})} \leq \frac{p_c \beta Z_h + p_e Z_h}{p_c \beta Z_h} = 1 + \frac{p_e}{\beta p_c}$. □

Furthermore, we have the following theorem if the prediction of Z_h is inaccurate.

Theorem 3. *If there is an error* ϵ *in the prediction of* Z_h*, that is,* $Z_h = (1 + \epsilon) Q(V, 0.95)$*, then we have the following inequality:*

$$\frac{\mathcal{A}(\mathcal{P})}{OPT(\mathcal{P})} \leq \begin{cases} 1 + \frac{p_e(1+\epsilon)}{\beta p_c}, & if\ \epsilon \geq 0 \\ 1 + \frac{p_e \sum_{k=1}^{K} B_k (1+\epsilon)}{\beta p_c Z_h}, & if\ \epsilon < 0 \end{cases} \tag{5}$$

Proof. Due to the space limitation, we omit the proof. □

C. Miss Ratio Predictor

The above algorithm has theoretical guarantees when the miss ratio is constant, which in practice varies depending on

Algorithm 2: Online Traffic Allocation

```
Input: $\{Gear_k\}$, $Z_h$, $Z_l$, $F$
for $k \in [1, K]$ and $n \in [0, N]$ do
    Initialize $Gear_k[n].usage \leftarrow 0$
When domain name traffic $\{h_t^d\}$ arrive at interval $t$
$v_t \leftarrow \sum_d h_t^d$, $used \leftarrow \varnothing$, $Z_h^{'} \leftarrow Z_h$, $Z_l^{'} \leftarrow \sum_k Bl_k$
if $v_t > Z_h^{'}$ then
    $used \leftarrow \{Gear_k[N] | k \in [1, K]\}$, $\{v_t^k\} \leftarrow \{Bl_k\}$
    Sort $used$ in ascending order by $usage$
    for $u \in used$ and $\sum_k v_t^k < v_t$ do
        $k \leftarrow u.id$, $v_t^k \leftarrow \min(v_t, B_k)$, $v_t \leftarrow v_t - v_t^k$
        for $n \in [0, N]$ and $Gear_k[n].value < v_t^k$ do
            $Gear_k[n].usage \leftarrow Gear_k[n].usage + 1$
else if $Z_l^{'} < v_t < Z_h^{'}$ then
    $index \leftarrow 0$, $notFinished \leftarrow True$
    while $notFinished$ do
        $used \leftarrow \{Gear_k[index] | k \in [1, K]\}$
        Sort $used$ in ascending order by $usage$
        for $u \in used$ and $v_t > 0$ do
            $k \leftarrow u.id$, $v_t^k \leftarrow v_t^k + \min(v_t, u.size)$
            $v_t \leftarrow v_t - \min(v_t, u.size)$
            $u.usage \leftarrow u.usage + 1$
        $index \leftarrow index + 1$
else if $v_t < Z_l^{'}$ then
    for $k \in [1, K]$ and $v_t > 0$ do
        $v_t^k \leftarrow v_t * \frac{Bl_k}{Z_l^{'}}$
        $Gear_k[0].usage \leftarrow Gear_k[0].usage + 1$
for $k \in [1, K]$ and $n \in [0, N]$ do
    if $Gear_k[n] > F$ then
        $Bl_k \leftarrow Gear_k[n].value$
return $\{v_t^k\}$
```

time and allocation policies. Therefore we design a predictor to estimate the midgress traffic a traffic allocation policy may generate, and to guide the policy selector. Specifically, the predictor will predict the miss ratio of each domain's requests on each edge server, *i.e.*, $\beta_t^{k,d}$, based on the historical traffic allocation information, thereby calculating the midgress traffic.

Different from the 3C's (cold, conflict, capacity) model of CPU cache proposed in [10], cache misses in CDNs are mainly caused by *cold miss* and *capacity miss*. A cold miss occurs when a file is requested for the first time in a cache. A capacity miss occurs when a requested file is evicted due to too many different files entering the cache since the last reference. We analyze these two kinds of misses separately. The miss ratio of a domain d's traffic sequence at cache size C is the sum of cold miss ratio mr_c^d and capacity miss ratio $mr_b^d(C)$.

Cold Miss. We use mr_c to represent the cold miss ratio. For a request sequence ρ of length l, let a_i denote the number of files requested i times in ρ that have never been requested in the cache before. Therefore, the cold miss ratio of sequence ρ in an empty cache is $\sum_i a_i/l$. Define $f_i = i * a_i/l$ as the probability that the file request frequency is i. If the request of ρ is randomly assigned to another empty cache with probability γ, we have the following lemma.

Lemma 1. $mr_c(\gamma) = \sum_i f_i \cdot \frac{1-(1-\gamma)^i}{i\gamma}$.

Proof. Please refer to Appendix B. □

The cache on the edge server may be non-empty due to previous traffic. Let ρ' denote the previous request sequence, $N_{i,j}$ denote the number of files that appear j times in the sequence ρ' and appear i times in the current request sequence ρ. If requests in sequences ρ' and ρ are allocated to the cache server with probability γ, we have the following lemma.

Lemma 2. $f_i = i \sum_j \frac{N_{i,j}(1-\gamma)^j}{l}$.

Proof. For each file in ρ' that is requested j times, the probability that it is not allocated to the cache server is $(1-\gamma)^j$. Thus, $a_i = \sum_j N_{i,j}(1-\gamma)^j$. □

In the process of online traffic allocation, we may need to adjust γ every once in a while. Therefore, we define the parameter ω_r for the file r, which represents the probability that file r has not appeared in the cache before. Suppose interval t's request sequence ρ_t is allocated to the cache server with probability γ_t. If file r is requested j times in ρ_t, ω_r is updated to $\omega_r \cdot (1-\gamma_t)^j$ and $a_i = \sum_{r \in R_i} \omega_r$ (R_i is the set of files requested i times). By combining Lemma 1 and Lemma 2 together, we have the following theorem.

Theorem 4. *If requests in sequence ρ_t at interval t are allocated with probability γ_t to a non-empty cache, the cold miss ratio of ρ_t is $\sum_i \sum_{r \in R_i} \omega_r[1-(1-\gamma_t)^i]/(\gamma_t l)$.*

In practical CDN systems, we find that if the time interval is divided into 5 minutes, the probability f_i of the file request frequency of adjacent intervals is similar. Thus, the probability distribution $\mathcal{F} = \{f_i\}$ of the last interval can be used for predicting the cold miss ratio of the current interval.

Capacity Miss. We denote the capacity miss ratio by mr_b and design a traffic descriptor to calculate it. The traffic descriptor is a probability distribution of a reuse distance [11] for domain traffic. A traffic descriptor can be represented by a tuple $Td = \langle l, p^r(x) \rangle$. l is the length of request sequence. $p^r(x)$ is the probability distribution that a reuse sequence of the domain traffic has x unique files. The capacity miss ratio at cache size C can be calculated by $mr_b(C) = \sum_{x>C} p^r(x)$.

Let $Td_1 = \langle l_1, p_1^r(x) \rangle$ be the traffic descriptors of a domain traffic sequence ρ_1. If requests in sequences ρ_1 are allocated to the cache server with probability γ, the new traffic descriptor can be calculated using the operation scaling ($\odot$), *i.e.*, $Td = \gamma \odot Td_1 = \langle \gamma \cdot l_1, p_1^r(x) \rangle$.

Let $Td_1 = \langle l_1, p_1^r(x) \rangle$ and $Td_2 = \langle l_2, p_2^r(x) \rangle$ be the traffic descriptors of two domain traffic sequences. The capacity miss ratio of a traffic mix at cache size C can be computed by operation addition ($\oplus$), whose rules are $mr_b(C) = Td_1 \oplus Td_2 = [l_1 \sum_{x > \frac{l_1 \cdot C}{l_1+l_2}} p_1^r(x) + l_2 \sum_{x > \frac{l_2 \cdot C}{l_1+l_2}} p_2^r(x)]/(l_1+l_2)$.

By combining operation addition and operation scaling, we can compute the miss ratio for complex domain traffic mixes. For example, assume half of requests for domain d_1 and all requests for domain d_2 are allocated to an edge server with cache size C. The descriptors of d_1 and d_2 are denoted by Td_1 and Td_2. We have the capacity miss ratio $mr_b(C) = (1/2 \odot Td_1) \oplus Td_2$. Then the total miss ratio of mixed traffic is $mr(C) = [(1/2)l_1 mr_c^{d_1} + l_2 mr_c^{d_2}]/(l_1/2 + l_2) + mr_b(C)$.

D. Policy Selector

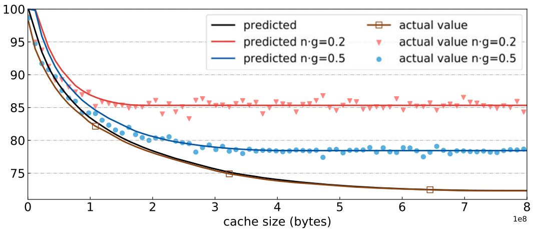


Fig. 4. The miss rate curves (MRCs) of different allocation ratio.

The policy selector selects the appropriate policy from the set of all the policies S for configuring the DNS server based on the information provided by the miss ratio predictor. As shown in Fig. 4, although the miss ratio predictor basically agrees with the simulation curve, there is still a certain prediction error when the cache size is small. This is because the error caused by the random allocation of requests is amplified. We provide a fault-tolerant policy selector in Algorithm 3. When the prediction error of the last interval is less than a certain threshold $e_{\max}$, the policy selector calculates the *estimated cost* of each policy in set S (Line 3). The estimated cost is defined as $p_e \Delta_e + p_c \Delta_c$, where Δ_e and Δ_c are the additional costs incurred in edge cost and midgress cost, respectively. Then the policy with the smallest estimated cost is selected as the output policy (Line 4-12). Otherwise, when the prediction error is large, the policy selector chooses to greedily allocate domain traffic (Line 13). Since a larger allocation ratio $n \cdot g$ of domain traffic means a lower miss ratio, Algorithm 3 takes the output v_t^k of the traffic scheduler as a soft constraint, and allocates the traffic of a domain name to the same edge server as much as possible (Line 14-18).

IV. Performance Evaluation

We conduct extensive evaluations in two steps: 1) evaluate the performance of our miss ratio predictor in a large-scale commercial CDN system and further validate it on two widely-used datasets: the production trace from Google [12] and the system benchmark of YCSB workloads from Yahoo [13]. We compare our miss ratio predictor with the footprint descriptor in [14] (in Sec. IV-A). 2) Applying the miss ratio predictor in our online traffic allocation framework `Iris`, we evaluate its performance in a large-scale commercial CDN system, and compare `Iris` with several state-of-the-art baselines, *e.g.*, Cascara [1], GIA [9], Load-Balance, and Baseline-Fit (in Sec. IV-B). We highlight our key findings as follows.

Algorithm 3: Policy Selection

```
   Input: $\{v_t^k\}$, $\{v_t^c\}$, $\{Bl_k\}$, $S$, $e_{max}$, $p_e$, $p_c$
1  $error \leftarrow 0$, $best \leftarrow \varnothing$
2  When domain name traffic $\{h_t^d\}$ arrive at interval $t$
3  if $error < e_{max}$ then
4      for $s \in S$ do
5          Calculate $\{\hat{v}_t^k\}$ from $s$ and $\{h_t^d\}$.
6          $vc_s \leftarrow$ MissRatioPredictor($s$), $\Delta_e \leftarrow 0$
7          $\Delta_c \leftarrow$ 95th of $\{v_t^c\}-$ 95th of $\{v_t^c\} \cup \{vc_s\}$
8          for $k \in [1, K]$ do
9              if $\hat{v}_t^k > Bl_k > v_t^k$ then
10                 $\Delta_e \leftarrow \Delta_e + (\hat{v}_t^k - Bl_k)$
11         $EstimatedCost_s \leftarrow p_e\Delta_e + p_c\Delta_c$
12     $best \leftarrow s \in S$ with the minimum $EstimatedCost_s$
13 else
14     Sort $\{h_t^d\}$ in descending order.
15     for $d \in D$ do
16         for $k \in [1, K]$ do
17             $n_t^{k,d} \leftarrow \min(\lfloor \frac{v_t^k}{h_t^d * g}, \frac{1}{g})\rfloor$
18     $best \leftarrow \{n_t^{k,d}\}$
19 Update the miss ratio predictor and the value of $error$.
20 return $best$
```

- Our missing rate predictor achieves a prediction with an average error of 1.8% when considering domain traffic mixing, 59.6% lower than the baseline algorithm.
- With accurate predictions, our online traffic allocation framework `Iris` reduces bandwidth costs by an average of around 8.149%, which means about $10 million in annual operating cost savings in CDNs.
- In the case of inaccurate predictions, `Iris` outperforms other algorithms within a certain error range.

A. Applying Miss Ratio Predictor in Commercial CDNs

Evaluation Setup. To perform our evaluation, we collect production traces from a large-scale commercial CDN system's servers from a metro area serving traffic for 22 domains over a period of a month. There are over a million requests every five minutes in the production traces. We set five minutes as a time interval. To obtain statistics based on historical data, we use two intervals of request information as a warm-up and predict the missing ratio for the next 5 hours. We compare the error of our miss ratio predictor (MRP) with the footprint descriptor (FD) proposed in [14].

Overall Result. We first select a domain of video type and a domain of image type respectively to conduct experiments on predicting the miss ratio of a single domain. We then conduct experiments with a mix of two domains and a mix of all 22 domains under a given allocation policy. Table I shows the prediction errors of MRP and FD under various traffic allocation strategies. We make the following observations.

TABLE I
PERFORMANCE OF PREDICTORS

Domain traffic mixes	Average error of FD	Average error of MRP
image	1.513%	1.239%
video	1.131%	1.825%
(1/2) video	3.686%	1.476%
image + (1/2) video	5.140%	**1.335%**
(1/2) all domains	4.471%	**1.804%**

First, when it comes to scaling and addition operations of domain traffic, the prediction error of MRP is significantly lower than that of FD. If we allocate all domains to an edge server with probability 1/2, the average prediction error of MRP is 1.8%, which is 59.6% more accurate than FD. Second, as for the results for a single domain, the prediction error of MRP for the image domain is slightly smaller than that of FD, while the prediction error of the video domain is larger than that of FD. This is because the user's access to video data has a strong burstiness, and the difference in the probability distribution of access frequency between adjacent intervals may be larger than that of other domains.

Impact of Cache Size. Fig. 5 demonstrates the impact of cache sizes on MRP and FD. As expected, in the cases of various domain combinations, the capacity miss ratio decreases as the cache size increases, resulting in a decrease in the total miss ratio. The prediction accuracy of MRP and FD decreases slightly with the increase in cache size, but the error of MRP is always smaller than that of FD. When the cache is larger than 10^9 bytes, the misses of user requests mainly come from cold misses, which still have a certain error in the prediction due to the unknown of future user request patterns.

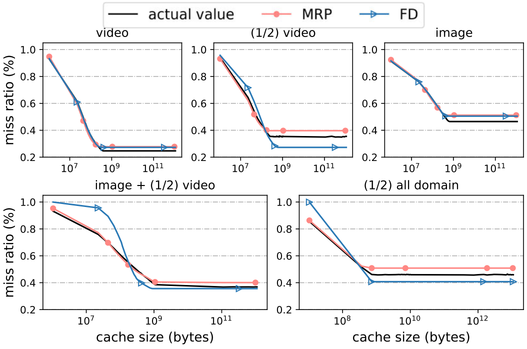


Fig. 5. The miss rate curves (MRCs) of domains on our CDN dataset.

Further Validation. We further conduct experiments with MRP and FD on Google and YCSB datasets. Since there is no domain information in these two traces, we treat Google trace and YCSB trace as one domain traffic each. Fig. 6 illustrates that the predictions of MRP are closer to the actual value than those of FD in most cases. Note that the request sequence patterns of these two traces are totally different. Requests for the same file in Google's trace usually arrive consecutively, while in YCSB they prefer to arrive independently [15]. When the cache size is less than 10^9 bytes, the request miss is mainly caused by the capacity miss. The prediction of MRP for the capacity missing ratio is based on the reuse distance probability distribution in the recent intervals, so its prediction is more accurate for Google's access pattern than YCSB's. But Google's access patterns may cause requests for new files to be concentrated within an interval, thereby affecting MRP's prediction of cold miss ratios. Therefore, when the cache size is greater than 10^9 bytes and the request misses are mainly caused by cold misses, the prediction accuracy of MRP on the Google's trace is worse than that on the YCSB's trace.

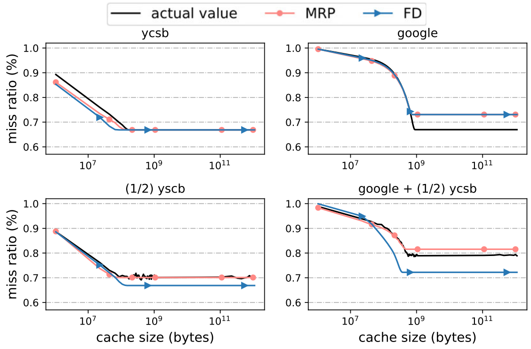


Fig. 6. The MRCs of domains on Google and YCSB datasets.

B. *Bandwidth Cost Optimization*

1) Evaluation Setup: We evaluate the performance of the traffic allocation framework `Iris` with the same production traces as Sec. IV-A. We choose a day as a charging period and every 5 minutes as an interval. Experiments are conducted independently on four weeks out of a month, and the results are reported on average. The cache size of an edge server is set so that the average traffic of one day is 70% of the cache size to simulate a typical CDN. We assume that every edge server in an area has equal bandwidth capacity and cache size. Due to the higher construction cost of the center server, its bandwidth unit price p_c is $1.5 \sim 3$ times p_e in industry.

2) Baselines: We compare `Iris` with significant baselines and their modifications:

Load-Balance (LB): It is the most widely used traffic allocation algorithm in industry [4]. When the domain traffic arrives, the traffic is allocated to an edge server according to the ratio between its bandwidth capacity and the total capacity. To meet the granularity requirements of DNS, we round the allocation ratio according to the granularity.

Cascara [1]: The key idea of Cascara is that when the domain traffic is greater than the predicted charging bandwidth C_f, select the capacity of edge servers below the charging bandwidth to load traffic according to the given priority. If no feasible solution exists, Cascara tends to increase C_f until a feasible solution is found. Similar to that in LB, we do the rounding in Cascara's allocation policy.

TABLE II
PERFORMANCE OF ALGORITHMS

Algorithms	LB	BF	Cascara	GIA	Iris
Edge cost	218.736	218.570	224.418	216.431	212.450
Midgress cost	176.732	154.207	169.508	170.868	150.789
Total cost	395.468	372.780	393.926	387.299	363.239
Cost reduction	—	5.736%	0.389%	2.065%	**8.149%**

GIA [9]: It is designed for situations where traffic is not divisible. GIA uses an offline algorithm to estimate the charging bandwidth of the current charging period. Then GIA greedily allocates the entire traffic to the edge servers with the estimated maximum uncharged capacity. In our experiments, we make granular domain traffic as an indivisible traffic flow.

Baseline-Fit (BF): It is an intuitive way to minimize the midgress cost, which always allocates the traffic of a domain to the same edge server as much as possible to improve the hit ratio of user requests on edge.

3) Experiment Results: We first compare the average bandwidth cost (in dollars) of each algorithm under the default configuration, and then explore the impacts of allocation granularity, prediction error and other factors on the performance.

Overall Result. Table II illustrates the average performance of algorithms under the data-trace from our large-scale commercial CDN collaborator. It should be noted that all algorithms are online. We define cost reduction as the percentage of bandwidth cost reduction of algorithm $\mathcal{A}$ compared to LB: $\frac{\texttt{Load-Balance}-\mathcal{A}}{\texttt{Load-Balance}} \times 100\%$. First of all, it can be observed that the cost reduction of Iris is reduced to 8.149%, which greatly reduces the bandwidth cost of traffic allocation compared with 5.736%, 0.389%, and 2.065% of the BF, Cascara, and GIA algorithms. This means that Iris can save more than \$800,000 per month in operating expenditure of CDNs over the LB currently used in industry. Secondly, as expected, except for Iris, BF can achieve a lower midgress cost, while other algorithms are insensitive to midgress traffic, and their midgress costs are higher. Thirdly, due to the full use of the free-of-charge intervals, Iris achieves the lowest edge cost compared to other algorithms, followed by the GIA algorithm. However, GIA lacks consideration of the trade-off between edge cost and midgress cost, resulting in high midgress cost, as does the Cascara algorithm.

Impact of the Number of Edge Servers. We keep the total bandwidth capacity of the edge servers constant and explore the impact of the number of edge servers K on the bandwidth costs. Each edge server has equal bandwidth capacity. Fig. 7 shows that Iris can effectively utilize the increased number of edge servers to reduce edge costs. When $K = 1$, all user requests can only be allocated to one edge server, so the bandwidth costs of all algorithms are equal. With the increase in the number of edge servers, Iris makes good use of the free-of-charge interval by using the green zones of edge servers in turn. However, when the number of edge servers is increased to 8, the bandwidth capacity of each edge server is too small, resulting in few feasible allocation policies and a slight increase in the bandwidth cost.

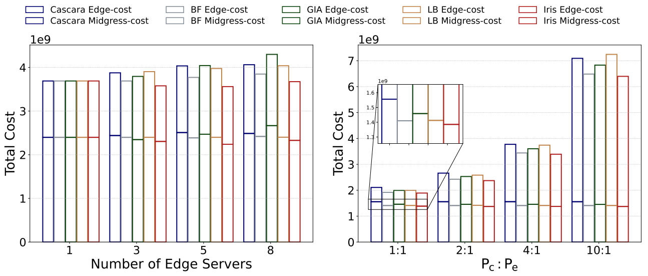


Fig. 7. Impact of K.

Fig. 8. Impact of unit price.

Impact of Unit Price. Fig. 8 explores the impact of the unit price of center and edge servers on bandwidth costs of all algorithms. When the ratio of the unit price of the center server to that of edge servers, the midgress costs increase accordingly. Iris always achieves the lowest total bandwidth cost through a trade-off between edge costs and midgress costs. When the ratio is $10 : 1$, Iris increases the edge cost by about 0.4%, and the total cost is reduced by 3.68%.

Impact of Allocation Granularity. Fig. 9 demonstrates the impact of allocation granularity on the performance of all algorithms when the charging period is six hours. As the allocation granularity increases, the performance of Iris has a certain degree of deterioration, mainly because the increment in allocation granularity will limit the allocation policy of our algorithm. When the granularity is less than $1/10$, the performance of Iris is always the best.

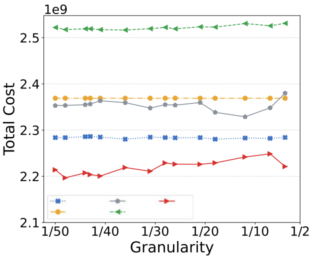


Fig. 9. Impact of granularity.

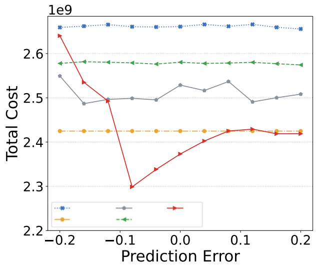


Fig. 10. Impact of prediction error.

Impact of Prediction Error. Note that both the GIA algorithm and Iris need to use historical data to predict the charging bandwidth as a key parameter. Therefore, we study the effect of the prediction error of charging bandwidth on algorithms. From Fig. 10, we can see that when the predicted charging bandwidth is in the range of [90%, 110%] of the accurate value, Iris yields the best performance among all algorithms. An interesting phenomenon is that better performance can be obtained if the sum of the red lines of edge servers in Iris is set to 92% of the predicted charging bandwidth. However, the curve on the left of 92% is steeper than the curve on the right, which means that under this scheme, if the predicted charging bandwidth is lower than expected, the performance of Iris may drop sharply. Therefore, to improve the fault tolerance rate and avoid excessive use of the red zones of edge servers, Iris conservatively sets the sum of the red lines of edge servers as the predicted charging bandwidth.

Impact of Bandwidth Capacity. In Fig. 11, we show the results of algorithms when the edge servers' bandwidth capacity configuration varies. The right half of Fig. 11 illustrates that when the bandwidth capacity of edge servers is doubled, the bandwidth costs of Iris, Cascara, and GIA are reduced. Iris is relatively stable in the two capacity ratio of edge servers, while the GIA fluctuates greatly. The reason is that the GIA algorithm is aimed at the situation where the domain traffic is indivisible, so if some edge servers are large enough to accommodate all the domain traffic, the algorithm works well, otherwise the cost may increase.

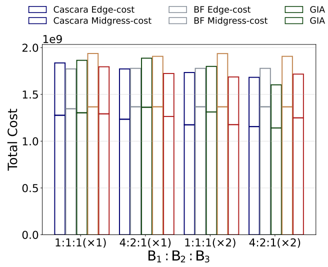


Fig. 11. Impact of capacity.

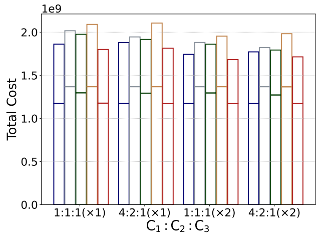


Fig. 12. Impact of cache size.

Impact of Cache Size. Comparing the right half of Fig. 12 with the left half, it can be found that when the cache size is doubled, the midgress cost of algorithms decreases significantly, and Iris always performs the best. When the cache size ratio of edge servers is 4:2:1, the bandwidth cost of algorithms increases slightly. The reason is that some requests are allocated to edge servers with smaller caches, resulting in more capacity misses and increased the midgress cost.

V. Related Works

We here give a survey of the mostly closed work in the literature on traffic allocation in CDNs.

Optimization of edge cost. In industry, the most widely used traffic allocation algorithm in CDNs is load balancing [4], which only focuses on ensuring that servers are not overloaded. Under the 95th charging scheme, an intuitive way to optimize bandwidth costs is to accumulate requests to certain time intervals and process them together so that the traffic bandwidth at most intervals is as low as possible [16]. Due to the difference in traffic peak times between regions with large time zone differences, Laoutaris *et al.* [17] propose to transfer data by taking advantage of already-paid-for off-peak bandwidth resulting. However, these solutions are only suitable for delay-tolerant backup data transmission. When user requests need to be allocated and satisfied by CDNs in real time, the optimization problem for 95th charging has been proved by Goldenberg *et al.* [9] to be *NP-hard*. Patrick *et al.* [18] design a scalable decentralized traffic allocation system DONAR to avoid transmission delays and optimize bandwidth costs at the same time. Paper [19] proposes to use the Shapley [20] value to guide the allocation of traffic bandwidth. Jalaparti *et al.* [21] model percentile-based usage costs as a compact set of linear inequalities and use the average of the top 10% link costs to estimate the edge cost. But Singh *et al.* [1] find that the correlation coefficient is weak in some links and design *Cascara* to reduce the percentile costs. Chen *et al.* [8] further consider the traffic allocation problem under the allocation granularity and deviation caused by the DNS mechanism.

Prediction and optimization of midgress cost. There has been a significant amount of research on cache management policies to reduce the midgress cost in CDNs by minimizing cache miss ratio, including [22]–[25]. These works on cache management are complementary techniques to traffic allocation. Using a midgress-sensitive approach to traffic allocation provides additional benefits than using cache management alone. By modeling the caching requirements of CDNs' traffic and predicting the best way to assign traffic, we can improve caching efficiency and provide an acceptable miss ratio at a reasonable bandwidth cost [14]. Mattson *et al.* [11] first propose a caching model based on stack distance, which is useful to compute miss ratio curves that plot the cache hit ratio as a function of cache size. Subsequent studies [26]–[28] have improved the time and space overhead of the algorithm in [11]. In the context of memory caching, Chandra *et al.* [29] propose three cache composition models to predict the impact on cache hit ratio when two non-overlapping applications run together in a shared cache. Hu *et al.* [30] develop a kinetic model of LRU cache when multiple traces share the cache, based on the average eviction time (AET). Sundarrajan *et al.* [14] design a traffic descriptor that supports a wider range of combinations of traffic classes, including addition and scaling. Based on the traffic descriptor, they design a traffic allocation algorithm that minimizes midgress cost [4]. However, their method can not work due to the nonlinearity of the 95th charging model.

VI. Conclusion

In this work, we study the bandwidth cost optimization problem for percentile charging in CDNs. We first provide an insight that previous algorithms ignore the trade-off between edge cost and midgress cost in CDNs, which can be optimized by predicting missing ratios when allocating traffic. Inspired by this, we develop a midgress-sensitive online traffic allocation framework, Iris, that achieves the prediction error of miss ratio within 1.804% and cost savings within 8.149% compared to the state-of-the-art baselines. We conduct comprehensive experiments on real data-trace from a global commercial CDN system. Iris significantly performs well even after increasing the adverse effects of prediction error and allocation granularity, potentially saving millions of dollars monthly for our commercial CDN collaborator.

Acknowledgements

This work was supported is partially supported by the National Key R&D Program of China under Grant 2021ZD0110400, Huawei Cloud Computing Technologies Co.

Ltd, NSFC under Grants 62132009 and 62132018, and the Fundamental Research Funds for the Central Universities at China. Haisheng Tan is the corresponding author (Email: hstan@ustc.edu.cn).

Appendix A
Proof of Theorem 1

Proof. The proof is by reduction from the *NP-complete Set-Partition Problem*, which partitions an array of numbers into two subsets such that the sum of each of these two subsets is the same. Let $S = \{s_1, s_2, \cdots, s_K\}$ be the set of numbers. Set $m = \sum_{i=1}^{K} s_i$. Then consider a CDN system with K edge servers whose bandwidth capacity is equal to the elements in S ($\forall i \in [1, K], B_i = s_i$) and a traffic sequence $V = \langle 0, 0, \cdots, \frac{m}{2}, \frac{m}{2} \rangle$. V represents the traffic of 20 intervals. The first 18 values of V are all 0, and the last two values are half the sum of the elements in set S.

Since the miss ratio is a constant β, the midgress traffic for the last two intervals is $\frac{m\beta}{2}$ regardless of allocation policies. Under 95th charging scheme, the midgress cost is $\frac{m\beta p_c}{2}$. It can be deduced that the Set-Partition Problem has solutions if and only if the given traffic allocation problem has a solution with a total cost $\frac{m\beta p_c}{2}$, that is, edge costs of 0. If we have a polynomial-time algorithm to the traffic allocation problem, we can solve the Set-Partition Problem by checking whether the cost returned by the algorithm is equal to $\frac{m\beta p_c}{2}$ or not. □

Appendix B
Proof of Lemma 1

Proof. In the preceding context, we give the definition of cold miss ratio, which is equal to $\sum_i a_i/l$ when we get the certain request sequence ρ on a cache. Now, we consider γ that represents the probability of each file being assigned, and it affects both the sequence length and the number of cold misses. For the new empty cache, its expectation of sequence length is $l^{'} = l * \gamma$. And for every file which is requested i times, the probability that it isn't assigned to the new cache is $(1-\gamma)^i$. So it is assigned with probability $1-(1-\gamma)^i$ and the number of cold misses is $a_i[1-(1-\gamma)^i]$. Thus, the cold miss ratio on the new empty cache:

$$mr_c(\gamma) = \sum_i a_i \cdot \frac{1-(1-\gamma)^i}{l'} = \sum_i \frac{a_i}{l} \cdot \frac{1-(1-\gamma)^i}{\gamma} = \sum_i f_i \cdot \frac{1-(1-\gamma)^i}{i\gamma}$$

□

References

[1] R. Singh, S. Agarwal, M. Calder, and P. Bahl, "Cost-effective cloud edge traffic engineering with cascara," in *Proc. of NSDI)*, 2021.

[2] Cisco Systems, "Cisco visual networking index: Forecast and trends, 20172022 white paper," https://davidellis.ca/wp-content/uploads/2019/05/cisco-vnifeb2019.pdf, 2019.

[3] Norton W.B, "Transit tactic-gaming the 95th percentile," 2014, http://drpeering.net/ Accessed 2015.

[4] A. Sundarrajan, M. Kasbekar, R. K. Sitaraman, and S. Shukla, "Midgress-aware traffic provisioning for content delivery," in *Proc. of USENIX ATC*, 2020.

[5] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, pp. 202–208, 1985.

[6] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE TPAMI*, vol. 39, pp. 2298–2304, 2016.

[7] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[8] H. Chen, H. Zhan, H. Tan, H. Xu, W. Shan, S. Chen, and X.-Y. Li, "Online traffic allocation based on percentile charging for practical cdns," in *Proc. of IEEE/ACM IWQoS*, 2022.

[9] D. K. Goldenberg, L. Qiuy, H. Xie, Y. R. Yang, and Y. Zhang, "Optimizing cost and performance for multihoming," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 79–92, 2004.

[10] M. D. Hill and A. J. Smith, "Evaluating associativity in cpu caches," *IEEE Transactions on Computers*, vol. 38, no. 12, pp. 1612–1630, 1989.

[11] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems journal*, vol. 9, no. 2, pp. 78–117, 1970.

[12] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, vol. 1, 2011.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proc. of SoCC*, 2010.

[14] A. Sundarrajan, M. Feng, M. Kasbekar, and R. K. Sitaraman, "Footprint descriptors: Theory and practice of cache provisioning in a global cdn," in *Proc. of CoNEXT*, 2017.

[15] C. Zhang, H. Tan, G. Li, Z. Han, S. H.-C. Jiang, and X.-Y. Li, "Online file caching in latency-sensitive systems with delayed hits and bypassing," in *Proc. of IEEE INFOCOM*. IEEE, 2022.

[16] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram, "Delay tolerant bulk data transfers on the internet," in *Proc. of the International Joint Conference on Measurement and modeling of Computer Systems*, 2009, pp. 229–238.

[17] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *Proc. of the ACM SIGCOMM*, 2011.

[18] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "Donar: decentralized server selection for cloud services," in *Proc. of ACM SIGCOMM*, 2010, pp. 231–242.

[19] R. G. Clegg, R. Landa, J. a. T. Araújo, E. Mykoniati, D. Griffin, and M. Rio, "Tardis: Stably shifting traffic in space and time," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, p. 593594, 2014.

[20] R. Stanojevic, N. Laoutaris, and P. Rodriguez, "On economic heavy hitters: Shapley value analysis of 95th-percentile pricing," in *Proc. of IMC*, 2010.

[21] V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache, "Dynamic pricing and traffic engineering for timely inter-datacenter transfers," in *Proc. of ACM SIGCOMM*, 2016.

[22] X. Hu, X. Wang, Y. Li, L. Zhou, Y. Luo, C. Ding, S. Jiang, and Z. Wang, "Lama: Optimized locality-aware memory allocation for key-value cache," in *Proc. of USENIX ATC)*, 2015.

[23] N. Beckmann, H. Chen, and A. Cidon, "Lhd: Improving cache hit rate by maximizing hit density," in *Proc. of NSDI*, 2018.

[24] 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.

[25] Z. Song, D. S. Berger, K. Li, A. Shaikh, W. Lloyd, S. Ghorbani, C. Kim, A. Akella, A. Krishnamurthy, E. Witchel *et al.*, "Learning relaxed belady for content distribution network caching," in *Proc. of NSDI*, 2020.

[26] G. Almási, C. Caşcaval, and D. A. Padua, "Calculating stack distances efficiently," in *Proc. of the workshop on Memory system performance*, 2002.

[27] C. A. Waldspurger, N. Park, A. Garthwaite, and I. Ahmad, "Efficient mrc construction with shards," in *Proc. of NSDI*, 2015.

[28] J. Wires, S. Ingram, Z. Drudi, N. J. Harvey, and A. Warfield, "Characterizing storage workloads with counter stacks," in *Proc. of OSDI*, 2014.

[29] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *Proc. of HPCA*. IEEE, 2005.

[30] X. Hu, X. Wang, L. Zhou, Y. Luo, C. Ding, and Z. Wang, "Kinetic modeling of data eviction in cache," in *Proc. of USENIX ATC*, 2016.