Midgress-aware traffic provisioning for content delivery

Aditya Sundarrajan, *University of Massachusetts Amherst;* Mangesh Kasbekar, *Akamai Technologies;* Ramesh K. Sitaraman, *University of Massachusetts Amherst & Akamai Technologies;* Samta Shukla, *CVS Health*

https://www.usenix.org/conference/atc20/presentation/sundarrajan

This paper is included in the Proceedings of the 2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the 2020 USENIX Annual Technical Conference is sponsored by USENIX.

Midgress-aware traffic provisioning for content delivery

Aditya Sundarrajan
UMass Amherst

Mangesh Kasbekar
Akamai Technologies

Ramesh K. Sitaraman
UMass Amherst
& Akamai Technologies

Samta Shukla
CVS Health

Abstract

Content delivery networks (CDNs) cache and deliver hundreds of trillions of user requests each day from hundreds of thousands of servers around the world. The traffic served by CDNs can be partitioned into hundreds of traffic classes, each with different user access patterns, popularity distributions, object sizes, and performance requirements. Midgress is the cache miss traffic between the CDN's servers and the content provider origins. A major goal of a CDN is to minimize its midgress, since higher midgress translates to higher bandwidth costs and increased user-perceived latency.

We propose algorithms that provision traffic classes to servers such that midgress is minimized. Using extensive traces from Akamai's CDN, we show that our midgress-aware traffic provisioning schemes can reduce midgress by nearly 20% in comparison with the midgress-unaware schemes currently in use. We also propose an efficient heuristic for traffic provisioning that achieves near-optimal midgress and is suitable for use in production settings. Further, we show how our algorithms can be extended to other settings that require minimum caching performance per traffic class and minimum content duplication for fault tolerance. Finally, our paper provides a strong case for implementing midgress-aware traffic provisioning in production CDNs.

1 Introduction

Content delivery networks (CDNs) carry more than 50% of all Internet traffic today [35] and that fraction is projected to increase over the coming years. Modern CDNs host a wide variety of content such as videos, software downloads, web pages, etc. that belong to hundreds of content providers. CDNs deploy hundreds of thousands of servers in clusters at the edge of the Internet to serve the hosted content to billions of end-users around the world. If the requested content is available in the edge server, a cache hit occurs and the end-user experiences a quicker response with lower latency. Otherwise, a cache miss occurs, and the edge server must fetch the content from the content provider's origin. A cache miss increases the user-perceived latency for a response and also increases the "midgress" traffic, which is the cache miss traffic between the CDN's edge servers and the content provider origins.

A CDN has many performance and cost objectives that must be optimized. Three important metrics are *origin offload* that is the amount of traffic offloaded from the origin servers, *end-user latency* that is the time between request and response for content as perceived by the end-user, and the *midgress bandwidth cost*[1] that is the cost of internal traffic in the CDN caused mainly due to cache misses at the edge servers. A metric that ties the three objectives together is the cache miss rate[2] which is the fraction of content bytes that were not present in the edge caches and needed to be fetched from origin. Smaller miss rate implies lesser cache miss traffic. Reduced cache miss traffic in turn implies increased origin offload, reduced end-user latency and reduced midgress bandwidth cost. Hence, minimizing the midgress, is a key performance objective from multiple perspectives.

Traffic classes. When users request content that is hosted on a CDN, the requests are classified into *traffic classes*. A traffic class is a collection of domains that host a specific type of content belonging to one or more content providers with similar requirements. For example, CNN videos and Apple iOS software downloads are each examples of a traffic class. Large CDNs host content that belong to hundreds of traffic classes. Recent work [66] has shown that traffic classes hosted on CDNs exhibit wide variations in popularity distributions, object size distributions and caching characteristics.

How CDNs serve content to users. Two interacting systems determine how content is served to users.

1) The *traffic provisioning system* decides which servers serve what fraction of each traffic class. Traffic provisioning

[1] The CDN also incurs a bandwidth cost for the "egress" traffic of content sent from the edge servers to the end-users. However, content providers pay the CDN for their egress traffic, while the midgress traffic is purely an overhead for the CDN operator that must be minimized.

[2] The miss rate metric that we use in this paper is sometimes called *byte* miss rate. An alternate definition is the (unweighted) fraction of *requests* that are cache misses and is less relevant for our work.

is performed periodically (say, once every few hours) as an *offline process* and uses the predicted user demand for the traffic classes and available server resources to produce an assignment of traffic classes to servers. Subsequently, each user request of each traffic class is routed [12] in *real-time* to a server that is provisioned to serve that traffic class[3].

2) Each CDN server has a cache that stores the content requested by users. Each server employs a *cache management system* that implements policies for managing the cache, such as an admission policy to decide what objects are cached and an eviction policy to decide what objects are evicted.

Minimizing midgress. The midgress bandwidth could cost tens of millions of dollars a year[4]. Thus, even a small reduction in midgress can be significant. Much of the prior work has focused on better cache management for reducing cache misses The past decades have seen research on numerous caching algorithms, such as Adapt-Size [4], Cliffhanger [15], SLRU [40], TLRU [23], S4LRU [34], CFLRU [59], ARC [53], LRU-S [65], LRU-K [56], and GDS [7]. However, the *complementary* problem of optimizing the traffic provisioning process to minimize midgress has not received much attention. In the current state-of-the-art, production CDNs assign traffic classes to servers with the goal of not overloading the servers, without explicitly minimizing midgress.

Our work shows that traffic provisioning in a midgress-aware manner can provide additional benefits to what can accrue from better cache management alone. Our traffic provisioning approach incorporates *both* traditional load balancing and the newer midgress considerations to minimize midgress traffic. *The main thesis of the paper is that by explicitly incorporating midgress considerations, it is possible to devise traffic provisioning schemes that minimize midgress traffic by nearly 20%, potentially resulting in millions of dollars of bandwidth cost savings.* Further, the midgress reduction due to better traffic provisioning is *complementary* to any improvements in cache management. As CDNs already implement traffic provisioning algorithms, albeit in a midgress-unaware manner, our contribution can be viewed as a drop-in replacement for an existing (midgress-unaware) traffic provisioning system.

Why be midgress-aware? "Midgress-aware" traffic provisioning algorithms explicitly incorporate cache miss traffic in addition to "balancing" the load. We illustrate the need for midgress awareness through a simple example. Consider two servers and three traffic classes. Each server has a cache size of 4 TB and sufficient capacity to serve all traffic classes. The three traffic classes have equal load of λ that need to be assigned to the two servers. The miss rate curves (MRCs) for the three traffic classes are as shown in Figure 1. The MRCs of traffic classes TC_1 and TC_3 flatten out quickly. This means that they require very little cache space to achieve the best possible performance. On the other hand, traffic class TC_2 has a slowly decreasing gradient. Thus, the miss rate of TC_2 keeps decreasing as more cache space is allocated to it.

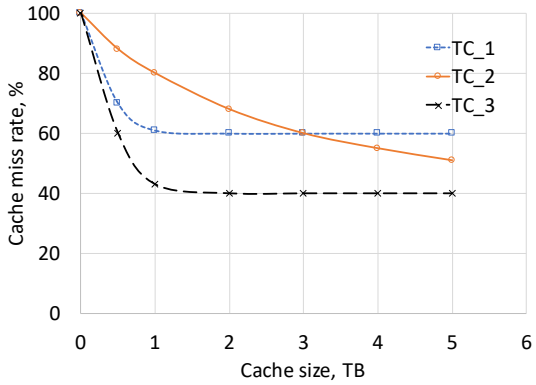

Figure 1: MRCs of traffic classes TC_1, TC_2 and TC_3.

Current traffic provisioning algorithms are *midgress-unaware* in that they only ensure that no server is overloaded. Such an algorithm could choose *any* assignment of traffic classes to servers, since any server has sufficient capacity to serve all classes, e.g., assigning TC_1 and TC_2 to server 1 and TC_3 to server 2 is one possible solution. More generally, any assignment with $(x+y+z)\times\lambda$ traffic to server 1 and $((1-x)+(1-y)+(1-x))\times\lambda$ traffic to server 2 is feasible, where x, y and $z \in [0,1]$, are the traffic fractions of TC_1, TC_2 and TC_3 respectively. Note that in this paper, we split the load of a traffic class by requests.

On the other hand, a midgress-aware algorithm would choose an assignment that minimizes the overall cache miss traffic from the two servers, while also ensuring that no server is overloaded. In the above example, assigning all of TC_1 and TC_3 to server 1 and all of TC_2 to server 2 would result in the least amount of cache miss traffic from the two servers. This is because TC_2 gets the largest cache space possible for its entire load and TC_1 and TC_3 get enough space to achieve the smallest cache miss rates.

1.1 Contributions

We make the following contributions.

1) We develop an optimization model for midgress-aware traffic provisioning that assigns traffic classes to servers in a manner that minimizes midgress traffic. The model is a non-convex mixed-integer linear program (MILP) that we solve using CPLEX. Our work is the first to explicitly model and minimize midgress in the traffic provisioning process. *Since a large CDN could incur a midgress of 10+ Tbps at a cost of $60+ million/year, even a small midgress reduction translates into large cost savings for the CDN.*

[3] The results of the traffic provisioning are used to create DNS records that can be resolved by the user in real-time using a DNS lookup [12].

[4] As a back-of-the-envelope calculation, a large CDN serving 50 Tbps of egress traffic at a 20% miss rate at the edge has a midgress traffic of 10 Tbps. The price of network bandwidth varies greatly throughout out the world. Though hard to estimate accurately, assuming a blended price of 50 cents per Mbps per month, midgress bandwidth costs 60 million dollars per year.

2) We apply our optimization solution to *metro-level traffic provisioning* where the traffic classes provisioned to server clusters within a metro area (e.g., NY city) are re-provisioned to minimize midgress. Metro-level traffic re-provisioning is a common operation, since the latency impact of moving a traffic classes across clusters within the same metro is likely minimal. Using extensive production traces from Akamai, we show that our midgress-aware traffic provisioning can reduce the midgress of a metro-area by 18.37% on average compared to midgress-unaware provisioning.

3) We also use our optimization solution for *cluster-level provisioning* where the traffic classes assigned to servers within a cluster are re-provisioned to minimize midgress. Cluster-level traffic (re-)provisioning is also a common operation since moving a traffic class across servers within the same cluster will likely not impact end-user latencies. Using production traces from Akamai's CDN, we show that cluster-level provisioning in conjunction with metro-level provisioning can reduce the midgress of a traffic class by 41.07% on average compared to midgress-unaware provisioning.

4) To be useful in practice, midgress-aware traffic provisioning has to be computationally efficient. We propose a midgress-aware heuristic called `local search` that is fast and near-optimal. The midgress achieved by `local search` was within 1.1% of optimal for both the metro-level and the cluster-level traffic provisioning. Further, in our experiments, `local search` completed in only 2 minutes, while finding the optimal took several hours.

5) We also show that our traffic provisioning algorithms are robust across different cache management policies and provide a midgress reduction in the range of 7.76% - 13.3%.

6) CDN operators often have to deal with additional constraints such as maintaining a certain level of traffic class redundancy or guaranteeing a minimum level of caching performance for traffic classes. We show how the optimization model for midgress-aware traffic provisioning and the heuristic algorithm, `local search`, can be extended to accommodate such constraints.

7) While the above results are for "shared" caches where a single unpartitioned cache is used to store objects from all traffic classes, we show that our traffic provisioning approach can be modified to work with "partitioned" caches where each traffic class is assigned a separate cache partition. We show that the midgress of partitioned caches can be reduced by more than 14% using our midgress-aware traffic provisioning approach, when compared to a midgress-unaware baseline.

1.2 Roadmap

The rest of the paper is organized as follows. In Section 2, we model midgress-aware traffic provisioning as a non-convex mixed-integer optimization problem. In Section 3, we propose a faster heuristic for midgress-aware traffic provisioning called `local search`, as well as a midgress-unaware baseline called `baseline fit`. In Section 4, we evaluate our optimization model and heuristics using extensive traces from Akamai's production CDN to empirically understand the midgress reduction achieved by our algorithms. In Section 5, we extend and evaluate our midgress-aware traffic provisioning algorithms to include other constraints such as minimum redundancy and maximum cache miss rates. Further, we extend our work to partitioned caches. We discuss some related work in Section 6 and conclude in Section 7.

2 Optimization model for traffic provisioning

We model traffic provisioning in a CDN as follows. We are given a set of N traffic classes. For each traffic class j, we are given the (predicted) amount of load of λ_j Gbps, $\forall j \in 1 \ldots N$. The predicted load for traffic provisioning is derived from historical load values for these classes by the CDN. Further, we are given M sites where the i^{th} site has a cache of size C_i TB and a capacity of T_i Gbps, $\forall i \in 1 \ldots M$. In cluster-level traffic provisioning, each site models a single CDN server within a cluster of M servers. In the more complex setting of metro-level traffic provisioning, we model an *entire* cluster as a single site within a metro area with M clusters. While not strictly accurate, we show that viewing the entire cluster as a single site in the metro-area setting is useful in practice. The capacity (resp. cache size) of each site is calculated as either the capacity (resp. cache size) of a single server in the former setting or as the aggregate capacity (resp. cache size) of the entire cluster in the latter setting. Henceforth, a site refers to a server in the cluster-level setting and a cluster in the metro-level setting.

The goal of traffic provisioning is to produce an assignment of traffic classes to sites, such that the total midgress across all the sites is minimized within the constraint that no site is assigned more load than its capacity. Note that a traffic class may be fractionally assigned across multiple sites, e.g., a traffic class with 10 Gbps of load can be assigned across two sites to host 7 Gpbs and 3 Gbps each of that class[5].

2.1 Modeling cache eviction and midgress

Given a site with an assignment of traffic classes, we need to model the miss traffic (i.e., midgress) that will result from serving those classes. The miss traffic is dependent on the cache management policies used by the sites. Nearly all production CDN caches use LRU (least-recently-used) variants as their eviction policy, since it is very efficient and achieves a comparable (byte) miss rate for typical CDN content traffic in comparison with other more complex eviction policies. For example, Akamai servers evict content using LRU, while admitting objects on second hit [47]. Production installations of the popular content caches Varnish [39] and NGINX [63]

[5] A CDN can implement such a fractionally-provisioned traffic class via a DNS mechanism that returns the ip address of the first site 70% and ip address of the second site 30% of the time.

also use LRU variants, as do recent academic work on content caching such as AdaptSize [4].

Production CDN servers also typically use a shared cache architecture where each server uses a single unpartitioned cache to serve all its traffic classes [66]. It is known that a partitioned cache that is sized in an optimal fashion can yield a greater reduction in midgress over a shared unpartitioned cache under the independent reference model (IRM) traffic assumptions [20]. However, in a production CDN, each server hosts a large number of traffic classes. Further, both the set of traffic classes hosted by a given server and the volume of traffic served per class by that server varies throughout the day. Thus, there is *significant* overhead involved in maintaining multiple cache partitions whose sizes must be dynamically varied throughout the day. The constant resizing of cache partitions could itself also lead to an increase in the midgress [61]. For these reasons, a shared unpartitioned cache is typically used by CDNs in practice.

In light of the above discussion, since our goal is to devise traffic provisioning algorithms to reduce midgress in production CDN settings, we develop a model for sites that use an LRU cache eviction policy with a shared cache architecture. But, later, we show empirically that our optimization model and algorithms produce a significant reduction in midgress, even if the CDN were to use other eviction policies (Section 4.3). Further, we show that our approach can also be easily extended to provide midgress reduction in a partitioned cache architecture (Section 5.3).

Eviction age equality. The eviction age of an object in cache is the difference between the time the object is evicted and the time that it was last accessed. In an LRU cache, at the time of access, the object goes to the head of the LRU list. Then, the eviction age of the object is the time for that object to move from the head to the tail of the LRU list and then get evicted. Thus, this time is about the same for all objects, when the size of an object is small with respect to the size of the cache. *We make the modeling assumption that the eviction age of all objects in cache are equal.* This assumption is also borne out in production caches and the common eviction age of the objects is logged as the eviction age of the cache.

The notion of eviction age can be extended to a traffic class by averaging the eviction age of all the requested objects from that traffic class. Since we model each object as having the same eviction age, all traffic classes assigned to a site share the same cache, and so they must have the same eviction age, which we also denote to be the eviction age of the cache. The eviction age of a cache has a direct relationship with the cache hit rate. Requests that have inter-arrival times less than or equal to the eviction age experience a cache hit and the rest experience a cache miss. So, for a given mix of traffic classes, as the cache size increases, the eviction age increases and so does the cache hit rate. Eviction age of a cache is similar to the concept of window size in [24]. Eviction age equality is crucial in our modeling of the midgress of traffic classes that share a single LRU cache.

2.2 Formulation of our optimization model

We now formulate our optimization model (referred to as OPT henceforth) for midgress-aware traffic provisioning.

Inputs of OPT. The input parameters used in the model are summarized in Table 1. We are given N traffic classes and M sites. The load λ_j of the j^{th} traffic class is given, for all $1 \leq j \leq N$. The cache size C_i and the capacity T_i of the i^{th} site is also given, for all $1 \leq i \leq M$. Further, for each traffic class, we are given the miss rate curve (MRC) and eviction age function as described below.

1) Miss rate curve (MRC), $\mathcal{M}_j(c)$. The MRC of a traffic class plots the cache miss rate as a function of cache size c. In this work, we assume that this function is convex (decreasing) which is generally true for stack-based algorithms [66]. As examples, MRC of two traffic classes, traffic class 2 and 14 (see Table 3) are shown in Figure 2.

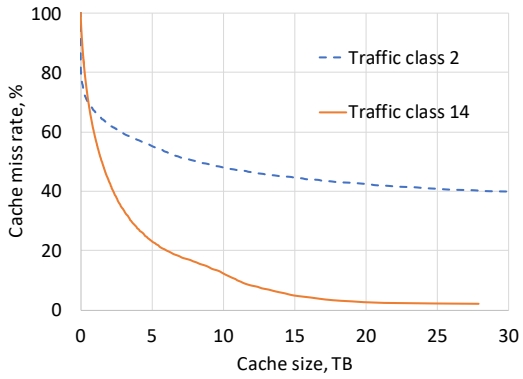

Figure 2: MRCs of two traffic classes.

From Figure 2, we can see that the MRCs are both convex. However, their gradients vary at different rates. Traffic class 2 has higher gradient at very small cache sizes but gradually flattens out as it reaches a cache space of 30 TB. Traffic class 14 on the other hand has a relatively high gradient until about 15 TB after which the MRC flattens out.

2) Eviction age function, $\mathcal{T}_j(c, \lambda)$. The eviction age function of a traffic class plots the eviction age at load λ as a function of the cache size c. The eviction age function also gives us information about *footprint pressure* of a traffic class, which is a relative measure of the amount of unique bytes accessed over a time period. A traffic class has high footprint pressure if a large number of unique bytes are accessed over a short time period. In this work, we assume that the eviction age function is convex (increasing) based on observations from production traces. The eviction age functions of two traffic classes, 2 and 14 (see Table 3) are shown in Figure 3.

From Figure 3, we can see that the eviction age functions are convex. As expected, at the given load, the eviction age increases with increase in cache size. Note that until about an eviction age of 2.1 days, traffic class 14 has higher footprint

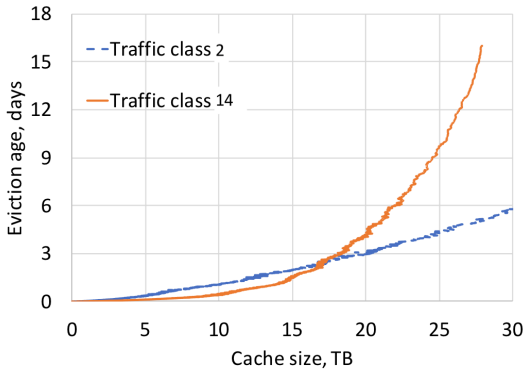

Figure 3: Eviction age functions of two traffic classes.

pressure when compared to traffic class 2, after which this behavior is flipped. Hence, if traffic classes 2 and 14 are assigned to the same site, traffic class 14 gets more cache space at smaller eviction ages ($\leq$ 2.1 days) due to higher footprint pressure and lesser cache space at larger eviction ages ($>$ 2.1 days) due to smaller footprint pressure.

To efficiently compute the MRC and eviction age function for every traffic class, we use a succinct space-time representation of the cacheability properties of a traffic class known as footprint descriptors [66].

We now briefly describe footprint descriptors.

Footprint descriptors. A footprint descriptor is a space-time representation of the caching properties of a traffic class. It is the joint probability distribution of the stack distance [51] (aka reuse distance) and the interarrival time distributions of a traffic class. A footprint descriptor can be used to determine the cache size and the eviction age that is required to achieve a certain cache hit rate, as a function of the traffic load. A footprint descriptor can also be used to determine the eviction age of a cache at different cache sizes and vice versa.

A reuse sequence is a sequence of requests where the first and the last request in the sequence is for the same object and that object is not requested anywhere else in that sequence. A simplified version of a footprint descriptor of a traffic class j is a tuple $< \lambda_j, P^r(s,t)) >$ where λ_j is the load of traffic class j and $P^r(s,t)$ is the reuse-sequence descriptor which is the joint probability distribution that a reuse sequence of the traffic class has s unique bytes and time duration t. Given a footprint descriptor, the miss rate curve at cache size s is defined as $MRC(s) = 1 - \sum_{s' \leq s} \sum_{t} P^r(s',t)$. The eviction age function $\mathcal{T}_j(s,\lambda_j)$, is computed from $P^r(s,t)$ by plotting the duration t as a function of unique bytes s at load λ_j.

Given the footprint descriptor of different traffic classes, the footprint descriptor of a traffic mix can be computed using the addition operator ($\oplus$) of the footprint descriptor calculus [66]. The crux of the addition operation is the convolution of the joint probability distribution, $P^r(s,t)$, of all the traffic classes, which can be efficiently computed using the Fast Fourier Transform algorithm. The MRC of the traffic mix can then be computed from the footprint descriptor of the traffic mix as described above. Note that the request characteristics of a traffic class could change slowly over time, requiring the footprint descriptor to be recomputed periodically.

Outputs of OPT. The output parameters of OPT are presented in Table 2. The primary output is x_{ij} that represents the fraction of traffic class j assigned to site i.

Notation	Description
N	Number of traffic classes
M	Number of sites
λ_j	Load of traffic class j
$\mathcal{M}_j(c_{ij})$	Miss rate of traffic class j at cache capacity c_{ij} in site i
$\mathcal{T}_j(c_{ij},\lambda_j)$	Eviction age of traffic class j at cache capacity c_{ij} and load λ_j in site i
C_i	Cache size of site i
T_i	Capacity of site i

Table 1: Input parameters of optimization model.

Notation	Description
c_{ij}	Cache size occupied by traffic class j in site i
ρ_i	Eviction age of site i and of traffic classes assigned to site i
x_{ij}	Fraction of $\lambda_j \in [0,1]$ assigned to site i

Table 2: Output parameters of optimization model.

Objective function. The objective of midgress-aware traffic provisioning is to assign the N traffic classes to the M sites such that the midgress traffic from all the sites is minimized as follows.

$$\text{min.} \sum_{i=1}^{M} \sum_{j=1}^{N} x_{ij} \lambda_j \mathcal{M}_j(c_{ij}) \tag{1}$$

Resource constraints. The first set of constraints are the cache size and the capacity constraints of each site.

$$\sum_{j=1}^{N} c_{ij} \leq C_i \quad \forall i = 1 \ldots M \tag{2}$$

$$\sum_{j=1}^{N} x_{ij} \lambda_j \leq T_i \quad \forall i = 1 \ldots M \tag{3}$$

The cache size constraint (Equation 2) states that the cache size occupied by all traffic classes assigned to all sites must not exceed the cache size of the site. The capacity constraint (Equation 3) states that the load of all traffic classes assigned to all sites should not exceed the capacity of the site.

Eviction age equality constraint. The eviction age function, $\mathcal{T}_j(c_{ij},\lambda_j)$ is defined at load λ_j for traffic class j. When traffic class j is assigned to site i, its load can be less than

or equal to λ_j due to fractional assignments. Let the load of traffic class j assigned to site i be $\lambda'_j \leq \lambda_j$. Then, the eviction age of traffic class j in site i is.

$$\mathcal{T}_j(c_{ij}, \lambda'_j) = \frac{\mathcal{T}_j(c_{ij}, \lambda_j)}{\lambda'_j/\lambda_j} = \frac{\mathcal{T}_j(c_{ij}, \lambda_j)}{x_{ij}} = \rho_i$$

The first equality is due to the fact that decreasing the load of a traffic class by a factor increases the eviction age of that class by the same factor, since eviction rate decreases by that factor. In the last equality, ρ_i is the eviction age of site i which is also the eviction age of all traffic classes that are assigned to site i. The eviction age equality constraint for all traffic classes at all sites is then given by

$$\mathcal{T}_j(c_{ij}, \lambda_j) = \rho_i x_{ij} \quad \forall j(i) = 1 \ldots N(M). \tag{4}$$

As previously discussed, the eviction age equality constraint in Equation 4 establishes the condition under which traffic classes assigned to site i share the cache.

Load assignment constraint. The load of a given traffic class can be fractionally assigned across sites. This means that for some traffic class j, 50% of the load λ_j could be assigned to site 1, 30% to site 2 and the remaining 20% to site 3, and so on. The load assignment constraint ensures that all the load of each traffic class is assigned to one or more sites.

$$\sum_{i=1}^{M} x_{ij} = 1 \quad \forall j = 1 \ldots N \tag{5}$$

Non-negativity constraints. The output parameters ρ_i, c_{ij} and x_{ij} should be non-negative.

$$\rho_i > 0 \quad \forall i = 1 \ldots M \tag{6}$$

$$c_{ij} \geq 0 \quad \forall j = 1 \ldots N \tag{7}$$

$$x_{ij} \in [0,1] \quad \forall j(i) = 1 \ldots N(M) \tag{8}$$

Together, Equations 1-8 constitute the optimization model for midgress-aware traffic provisioning OPT.

2.3 Solving the optimization model OPT

The complexity of solving the optimization model OPT proposed in Section 2.2 is evaluated as follows. The objective function (Equation 1) is biconvex since the load fraction x_{ij} is linear and the MRC $\mathcal{M}_j(c_{ij})$ is convex. Equations 2-3, 5-8 are affine constraints. The eviction age function $\mathcal{T}_j(c_{ij})$ is convex and the product term $\rho_i x_{ij}$ is bilinear. Equation 4 is a non-convex constraint because the feasible set defined by this constraint is non-convex. Overall, the optimization problem is non-convex and in general an NP-hard problem. We make a number of mathematical transformations to convert the optimization problem to a mixed integer linear program (MILP), which in turn can be solved efficiently using CPLEX.

3 Traffic provisioning heuristics

The optimization model OPT proposed in Section 2.2 is an NP-hard problem and it can take several hours for a solver to obtain the exact optimal solution. A faster but approximate solution is valuable for a large production CDN that has hundreds of traffic classes, 1000+ clusters with deployments in every major metro region of the world. To that end, we propose a traffic provisioning heuristic called `local search` that is fast and sufficiently accurate to be used in production. Intuitively, our traffic provisioning heuristic is a "hill climbing" solution for our optimization model in Section 2.2. We also consider a midgress-unaware traffic provisioning algorithm called `baseline fit` that we use as a baseline to evaluate the benefits of being midgress-aware. The `baseline fit` algorithm is similar to the midgress-unaware algorithms currently used in production settings.

3.1 Midgress-unaware baseline

The midgress-unaware traffic provisioning algorithm called `baseline fit` (see Algorithm 1) is based on consistent hashing, similar to the algorithms used in production settings [47]. The algorithm takes as input the set of N traffic classes and the set of M sites that are both hashed to points on a unit circle. The traffic classes are picked in a random order and assigned to sites as follows. Each traffic class j is assigned to the nearest site i on the unit circle in the clockwise direction. If the chosen site i does not have enough capacity to host the entire load λ_j, then a first fit algorithm is used, starting from the chosen site i, and continuing to subsequent sites on the unit circle in the clockwise direction, until all traffic is assigned. The key point to note is that `baseline fit` does not explicitly minimize the miss traffic, but rather it only ensures that no site gets more load than its capacity. That is, it produces a *feasible* solution for our model OPT by obeying Equations 2 - 8, but does not minimize midgress.

Algorithm 1 `Baseline fit` algorithm

Input: $N, M, \lambda_j, C_i, T_i$
Output: Fraction of traffic class j assigned to site i, $x_{ij}, \forall j(i) = 1 \ldots N(M)$

```
x_ij = 0
TC_set = set of all traffic classes arranged in random order
S_set = set of all sites hashed to a unit circle
for all j ∈ TC_set do
    i = Site chosen by consistent hashing
    if site i has remaining traffic capacity ≥ λ_j then
        x_ij += 1
    else
        Assign traffic fraction λ_j using first fit starting from site
        i on the unit circle
```

3.2 Midgress-aware local search

We propose a midgress-aware traffic provisioning algorithm called `local search` (see Algorithm 2) that uses a hill climbing approach to solve the optimization model OPT. It is designed to be fast, but may not always produce the optimal solution. The algorithm `local search` begins with a feasible assignment as determined by `baseline fit`. The algorithm operates in rounds where every traffic class is picked one at a time in each round. The traffic class that is picked is reassigned in small increments of a fraction δ ($0 < \delta < 1$) of its load to the server that minimizes the midgress objective while maintaining feasibility. If a round does not decrease the midgress traffic objective by at least a specified $\varepsilon << 1$, the algorithm stops and outputs the final assignment.

Note that `local search` could end up in a local optimum that isn't close to the global optimum. However, `local search` is efficient enough that it can be run multiple times (in parallel) with different starting points to improve a suboptimal solution. We discuss the running time of `local search` in Section 4.

Computing the midgress of a traffic assignment. The `local search` algorithm requires an efficient way to compute the midgress traffic of each site, given a traffic class assignment. A known technique for computing miss traffic of a site is footprint descriptor calculus [66]. Knowing the footprint descriptor of each traffic class that is assigned to a site, we use the calculus to efficiently derive the footprint descriptor for the traffic mix, that in turn provides the MRC of the traffic mix, from which we derive the midgress of the traffic mix.

Algorithm 2 `Local search` algorithm

Input: $N, M, \lambda_j, C_i, T_i$
Output: Fraction of traffic class j assigned to site i, $x_{ij}, \forall j(i) = 1 \ldots N(M)$

Get feasible assignment using `baseline fit` algorithm
TC_{set} = set of all traffic classes arranged in *random* order
S_{set} = set of all sites
while True **do**
    mg_{curr} = midgress of current assignment
    for all $j \in TC_{set}$ **do**
        $x_{ij} = 0 \quad \forall i = 1 \ldots M$
        $\lambda' = \lambda_j$
        while $\lambda' > 0$ **do**
            $S^j_{set} \subseteq S_{set}$ = set of all sites with remaining traffic capacity $\geq \delta\lambda_j$
            if $S^j_{set} \neq \emptyset$ **then**
                i = site in S^j_{set} that gives the lowest overall midgress after assigning TC j
                $x_{ij} \mathrel{+}= \delta$
            else
                Assign load $\delta\lambda_j$ using fractional first fit starting from a random site
            $\lambda' \mathrel{-}= \delta$
    mg_{new} = midgress of new assignment
    if $mg_{curr} - mg_{new} < \varepsilon$ **then**
        break

4 Experimental evaluation

Using production traces collected from a metro area of Akamai's CDN, we compare the midgress of OPT with that of `baseline fit` and `local search` in both metro-level and cluster-level traffic provisioning. We perform the evaluation in two steps: 1) We evaluate metro-level traffic provisioning by viewing each cluster as a site. The site is assumed to have cache size and capacity equal to the sum of the cache sizes and capacities of all servers in that cluster. The output of metro-level traffic provisioning is an assignment of traffic classes to clusters that minimizes the midgress of the metro area. 2) The output of metro-level traffic provisioning is the input to cluster-level traffic provisioning. We evaluate cluster-level traffic provisioning by assigning traffic classes to servers within a cluster to further minimize midgress.

Production traces. To perform our evaluation, we collect production traces from all Akamai CDN servers from a metro area serving traffic for 25 traffic classes over a period of 16 days. The characteristics of the traffic classes are listed in Table 3. From Table 3, we see that, in this metro area, 9 traffic classes serve web content, 11 traffic classes serve media content and the remaining 5 traffic classes serve software downloads. The traffic classes exhibit a wide variation in load (Gbps), arrival rate (requests/sec), content footprint (in unique bytes), and number of objects. The majority of the load is for media content at 47.3% followed by software downloads at 41.5% and web content at 11.2%. In terms of the unique bytes that are cached in the metro area, the majority is again for media content at 60.9%, followed by 25.6% for web content and 13.5% for software downloads.

Footprint descriptors described in [66] are periodically computed for all traffic classes on the production CDN. We use these footprint descriptors to compute the MRCs and the eviction age functions for the 25 traffic classes in Table 3, to be used as inputs to our traffic provisioning algorithms.

Evaluation setup. To evaluate the traffic provisioning algorithms, we simulate a small metro region with 10 clusters, each containing 10 servers[6]. The capacity of the metro region is set so that the average load is 70% of capacity to reflect the load-to-capacity ratio in a typical CDN. We evaluate the traffic provisioning algorithms at different cache sizes per cluster of 1 TB, 5 TB, 10 TB, 20 TB, 40 TB and 50 TB. For simplicity, we assume that every cluster in the metro area has equal capacity and cache size. Every server within a cluster is also assumed to have equal capacity and cache size.

OPT is solved using CPLEX as part of the GAMS modeling

[6] While a metro region in a large CDN typically has much larger server deployments, we simulate a scaled-down version to keep our experiments computationally tractable.

Traffic class id	Content type	Load (Gbps)	Arrival rate (req/s)	Unique bytes (TB)	Unique objects (million)
1	web	0.39	438.41	1.83	16.36
2	web	1.12	232.48	70.74	38.38
3	media	3.75	345.94	198.85	176.68
4	web	0.24	143.67	0.008	0.03
5	web	0.17	145.13	0.03	0.08
6	download	4.74	1338.91	28.16	19.55
7	web	0.30	851.73	6.21	70.23
8	web	0.58	1213.87	6.38	137.60
9	web	1.59	714.42	22.58	52.91
10	download	0.39	307.92	1.68	0.82
11	download	10.66	809.29	22.74	10.75
12	media	0.43	110.22	14.13	24.41
13	web	0.0013	136.32	0.04	3.58
14	media	7.54	93.01	30.55	2.90
15	media	7.22	89.28	30.14	2.86
16	media	6.04	75.14	30.38	2.89
17	media	0.37	139.23	12.41	26.59
18	web	2.12	935.76	83.42	93.54
19	media	0.35	134.87	24.48	25.12
20	download	1.36	276.63	3.12	2.07
21	media	0.08	9.94	7.31	6.43
22	media	0.90	214.53	43.48	77.90
23	media	0.44	48.28	28.53	26.83
24	media	0.38	78.09	35.25	55.06
25	download	6.99	1879.65	44.94	21.02

Table 3: Traffic class characteristics

language. We use a macOS machine with a 3 GHz Intel Xeon processor with 10 cores and 128 GB RAM for all our experiments. The GAMS program is set to run in parallel mode using 20 threads with a relative optimality gap of 1e-9 and a maximum run time of 40,000 s. Given the complexity of the optimization model, the GAMS program almost always runs for 40,000 s. At that point, the solver has converged to a solution that seldom changes and achieves a relative gap of under 5% at smaller cluster cache sizes less than or equal to 10TB and a relative gap of under 10% at larger cache sizes. A single run of `baseline fit` takes about 1 s and a single run of `local search` takes about 120 s.

4.1 Metro-level traffic provisioning

We evaluate OPT, `baseline fit`, and `local search` by computing the cache miss rate of the entire 10-cluster metro area for different cache sizes. These three algorithms each assign the set of 25 input traffic classes to the clusters in the metro. In the case of `baseline fit` and `local search`, we report the average cache miss rate of 100 runs, where each run considers the traffic classes in a random order. The 95% confidence intervals of the expected cache miss rates have a margin of error of less than 0.4%.

From Figure 4 we can see that OPT gives a 18.37% reduction in midgress on average compared to the midgress-unaware `baseline fit` algorithm. This is because OPT takes into account the impact on midgress while assigning traffic classes to clusters in the metro area. This significant improvement in midgress makes the case for implementing midgress-aware traffic provisioning algorithms in CDNs.

From Figure 4, we also see that `local search` performs quite well and gives a 15.44% reduction in midgress on average compared to `baseline fit`. `local search` also performs fairly well compared to OPT, with a modest 3.69% increase in midgress compared to OPT on average.

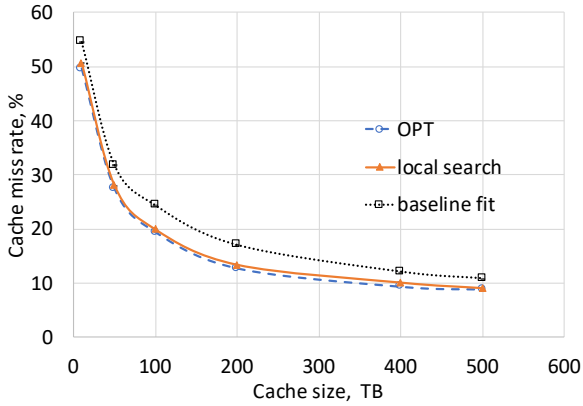

Figure 4: MRCs of OPT, `local search` and `baseline fit`.

4.1.1 How traffic provisioning impacts midgress

In Figure 5, we plot the cache miss rate of the 25 traffic classes when they are provisioned using OPT versus the midgress-unaware `baseline fit`, when the cumulative cache size of clusters in the metro area is 100TB. In addition, we also plot the average number of sites (across the 100 runs) that each traffic class is assigned to in Figure 6. From these figures, we see that OPT reduces the cache miss rate of 21 traffic classes when compared to `baseline fit`. In the case of traffic class 11, OPT results in almost 97% reduction in miss rate when compared to `baseline fit`. On the other hand, OPT increases the cache miss rate of four traffic classes, namely 4, 5, 13 and 19. By trading off the cache miss rates for these four traffic classes, OPT is able to reduce the overall midgress. But why does OPT choose this trade-off? There are three key insights that midgress-aware traffic provisioning takes into account to optimize midgress that `baseline fit` does not.

1) *In OPT's solution, traffic classes that have higher load, higher footprint pressure and greater MRC gradients get to occupy larger portions of the available cache space.* A traffic class has high footprint pressure if a large amount of unique content bytes is requested in a short period of time. This is true for traffic classes 11, 14, 15, 25 and 16 that account for 66.04% of the total load. OPT assigns traffic class 11 to two clusters because its load is greater than the capacity of a single cluster, resulting in that traffic class occupying 6 TB in one

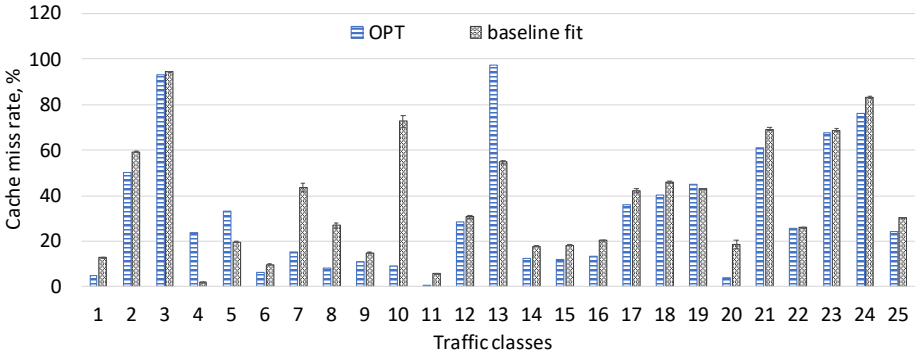

Figure 5: Average miss rate of each traffic class in a metro area of cache size 100 TB.

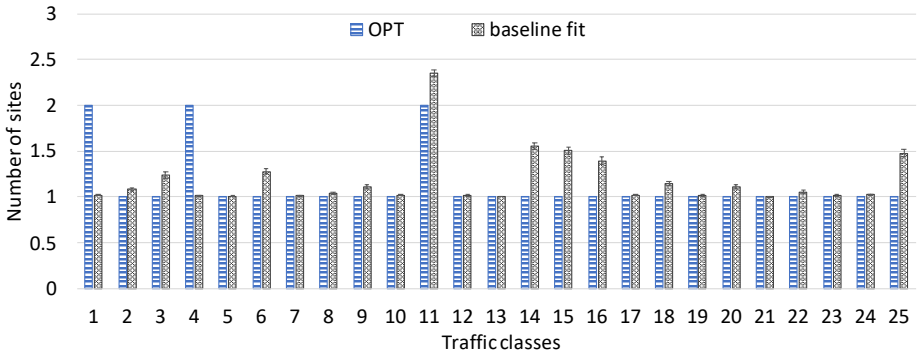

Figure 6: Average number of sites each traffic class is assigned to in a metro area of cache size 100 TB.

cluster with a miss rate of 0.5% and 7 TB in another cluster with a miss rate of nearly 0%. OPT also assigns an entire cluster each to traffic classes 14, 15, 25 and 16.

2) *OPT may split a traffic class and assigns it to multiple clusters if it has a relatively flat MRC.* This is true of traffic class 1 which has a relatively flat MRC and is assigned to two clusters. By reducing its footprint pressure in each of its assigned clusters, traffic class 1 is able to cede cache space to other traffic classes that are in more need.

3) *In OPT's solution, traffic classes that have lower footprint pressure occupy smaller portions of the available cache space.* This is true for traffic classes 4, 5 and 13. It also happens to be the case that these 3 traffic classes have very low load among the traffic classes considered. Both these factors render a higher cache miss rate relative to `baseline fit` that is midgress unaware. Note that low load alone does not indicate that it will occupy a smaller portion of the cache. For instance, traffic class 24 has moderate load but it has high footprint pressure and a greater MRC gradient, and ends up occupying 4.2 TB in one cluster.

4.2 Cluster-level traffic provisioning

The goal of cluster-level load balancing is to assign traffic classes to servers such that the midgress of the cluster is minimized. In our evaluation, we take the output of metro-level traffic provisioning from Section 4.1 that assigns traffic classes to each cluster and treat them as the inputs to cluster-level traffic provisioning. In this manner, we are able to understand the additional midgress reduction that is achievable by performing optimization at the cluster level, given that the metro level has already been optimized.

For cluster-level traffic provisioning, each traffic class defined at the metro-level is typically split into multiple finer-grained subclasses. The subclasses allow better allocation of traffic classes within a cluster. Traffic class 14 has very high load and hence was assigned to a cluster all by itself (Figure 6) by OPT at the metro-level. We considered that cluster for our evaluation of cluster-level traffic provisioning. Traffic class 14 consisted of 66 traffic subclasses that must be assigned to the 10 servers within a cluster, each server with a 1 TB cache.

OPT reduced the midgress for traffic class 14 by 31.26% after the metro-level optimization, when compared to the midgress achieved by `baseline fit`. Further, after using OPT for cluster-level provisioning, the midgress for traffic class 14 reduced further by 14.26%. In aggregate, the overall reduction of the midgress due to both provisioning steps of OPT is 41.07%, when compared to the baseline. Algorithm `local search` provided nearly as much reduction as OPT. For instance, `local search` provided a midgress reduction of 35.49%, compared to the baseline. However, `local search` was much faster and completed within 2 minutes, as opposed to the nearly 40,000 s (∼11 hours) taken by OPT.

4.3 Robustness to cache management policies

So far, we have developed traffic provisioning algorithms that model an LRU cache and evaluated the midgress reduction resulting when the sites also use LRU. The past decades have seen much academic research on numerous cache management algorithms that admit and evict objects using some combination of *recency* of access, *frequency* of access and object *size* to reduce cache miss rates (see Table 2 of [4]). We show that midgress-aware traffic provisioning algorithms proposed in this work, that model an LRU cache, achieve significant midgress reduction even when a CDN *does not* actually implement LRU at its sites.

We choose three typical algorithms from the literature for our evaluation. The first is an LRU variant called second-hit-LRU (or, SH-LRU) where the object is admitted to an LRU cache on second hit. The second is segmented LRU (SLRU) [40] that uses both recency and frequency for cache management. Finally, we implement the Greed-Dual-Size-Frequency (GDSF) [13] that uses all three of recency, frequency and size. Our evaluation uses the same cluster-level scenario as described in Section 4.2, where the goal is to assign the 66 traffic subclasses of traffic class 14 across 10 servers of size 1 TB each. First, we solve OPT that models LRU to get the optimal traffic class assignment across all servers within the cluster. The midgress of OPT's assignment is then computed by simulating the different cache management algorithms using the request traces of the subclasses. For comparison, we use the midgress-unaware `baseline fit` for traffic provisioning followed by a trace-based simulation of the different cache management algorithms to provide a baseline. When LRU cache management is used, OPT reduces the midgress by 13.3% when compared to `baseline fit`. In comparison, OPT reduces the midgress by 7.78%, 8.45% and 7.76% for SH-LRU, SLRU and GDSF respectively. The midgress reduction for the non-LRU algorithms is not as much as that for LRU. However, the midgress reductions for other algorithms are still quite robust and significant. The reason is that even when other factors are used for cache management decisions, most reasonable algorithms *still* use recency of access in a very significant way, and recency is well-captured in OPT. It is plausible that OPT can be reformulated to capture other cache management policies besides LRU and such an extension is a topic for future work.

5 Extending midgress-aware provisioning

We propose two extensions of midgress-aware traffic provisioning that address constraints in production settings. The first extension enforces a minimum number of sites that a traffic class must be assigned to and the second extension enforces a maximum cache miss rate per traffic class.

All results presented until now are for shared caches. While partitioned caches are not commonly used in production settings due to the overheads of dynamically resizing those partitions, there is increasing interest to implement and evaluate partitioned caches [1,5,9,14,16,21,25,36,43,44,46,62,67,69]. We propose a third extension to show that our traffic provisioning approach can reduce midgress in partitioned caches.

5.1 Minimum redundancy guarantee

Let M_j be the minimum number of sites that traffic class j should be assigned to. M_j is an integer $\in [1,M]$, where M is the total number of sites. Let y_{ij} be an indicator variable that is set to 1 when $x_{ij} > 0$ and 0 otherwise. Then, the load assignment constraint in Section 2.2 is appended to include the following minimum redundancy constraints.

$$y_{ij} = \lceil x_{ij} \rceil \quad \forall\, j = 1 \ldots N \tag{9}$$

$$\sum_{i=1}^{M} y_{ij} >= M_j \quad \forall\, j = 1 \ldots N \tag{10}$$

$$y_{ij} \in \{0,1\} \tag{11}$$

Equations 9 and 10 ensure that traffic class j is assigned to at least M_j sites. We call the modified optimization model OPT-M. The additional constraints are affine and they do not increase the complexity of the optimization problem. Both `local search` and `baseline fit` can also be modified to incorporate the redundancy constraint by simply ensuring that each traffic class j is assigned to at least M_j sites in each (re-)provisioning step.

Experimental evaluation. We measure the reduction in midgress by OPT-M and the modified `local search` when compared to the modified `baseline fit`. We use the same evaluation parameters as Section 4.1 where the cache size of each cluster in the metro area is 10 TB. We find that the cache miss rates increase with increase in redundancy for all three algorithms. We also find that the cache miss rate of `baseline fit` with minimum redundancy = 1 (resp. 2) is similar to the cache miss rate of `local search` and OPT-M with minimum redundancy 2 (resp. 3). This shows that midgress-aware traffic provisioning can provide the same midgress as `baseline fit` with added redundancy.

5.2 Maximum cache miss rate guarantee

Let MR_j be the maximum cache miss rate for traffic class j. Then, the optimization model in Section 2.2 can be extended to incorporate the maximum cache miss rate guarantee.

$$\sum_{i=1}^{M} x_{ij} m_j(c_{ij}) \leq MR_j \quad \forall j = 1 \ldots N \tag{12}$$

Equation 12 states that the average miss rate of traffic class j across all M sites should be at most MR_j. We call the modified optimization model, OPT-MR. Equation 12 is a biconvex constraint and doesn't increase the complexity of the problem.

We make two modifications to `local search`. First, `baseline fit` in the first step does not always provide a feasible solution when $MR_j < 100\%$. This is because `baseline fit` is midgress unaware. Hence, we start with all traffic classes being unassigned. Second, the re-provisioning step assigns a traffic class to a site only when the miss rate guarantees of all traffic classes assigned to that site are not violated.

Infeasible solutions. OPT-MR can be infeasible in cases where certain traffic classes fail to meet the maximum cache miss rate MR_j guarantee. For example consider a cache size of 10 TB. The lowest miss rate that traffic class 3 (Table 3) can possibly achieve at 10 TB is 91%. Hence, any maximum cache miss rate target less than 91% cannot be achieved.

Experimental evaluation. We choose traffic classes 13, 23 and 24 that have high miss rates in OPT and set their maximum cache miss rates to 70%. We evaluate the performance of metro-level traffic provisioning under the same conditions as in Section 4.1 where the cache size is 10 TB.

OPT-MR returns a feasible solution. The overall miss rate of the metro area is 20.04%, a 3.24% increase in midgress compared to OPT. In the process, three traffic classes experience a significant increase in their respective miss rates relative to OPT. The miss rate of traffic class 2 increases from 50.12% to 65.85%, of traffic class 17 from 36.11% to 54.2% and of traffic class 21 from 61.22% to 68.01%. This is because traffic classes 13 and 24 occupy more cache space with OPT-MR than they do in OPT, so they meet their miss rate guarantee, despite traffic class 13 having the lowest load and low footprint pressure, and traffic class 24 having low load.

We run the modified `local search` 100 times with different random orderings of the input traffic classes. `local search` returns a feasible solution 67% of the time indicating that its feasibility depends on the ordering of the traffic class inputs. For feasible assignments, `local search` has an average miss rate of 21.69%, about 8.23% more than that of OPT-MR. `baseline fit` is footprint-unaware and cannot guarantee cache miss rates.

5.3 Traffic provisioning in partitioned caches

In a partitioned cache, each traffic class is assigned to its own separate cache partition and each partition performs evictions independently. Production CDNs do not typically implement partitioned caches due to the significant overheads involved in implementing and dynamically maintaining the partitions. However, we show that our optimization model for midgress-aware traffic provisioning can be extended to work with partitioned caches.

Modeling and implementing traffic provisioning for partitioned caches. In partitioned caches, every traffic class occupies a separate partition with its own LRU list, assuming the LRU eviction policy. Thus, the eviction age of each traffic class assigned to the same cache can be different. Therefore, the optimization model for midgress-aware traffic provisioning for partitioned caches is the same as that of OPT (Section 2.2), minus the eviction age constraint. Hence, Equations 1-3 and 5-8 accurately model midgress minimization for partitioned caches. We call this modified model OPT-part.

We implement a baseline midgress-unaware algorithm called `baseline fit-part` for partitioned caches that is based on consistent hashing. The algorithm takes as input the set of N traffic classes and the set of M sites that are both hashed to points on a unit circle. Each traffic class j is assigned to the nearest site i on the unit circle in the clockwise direction. If the chosen site i does not have enough capacity to host the entire load λ_j, then a first fit algorithm is used, starting from the chosen site i, and continuing to subsequent sites on the circle in the clockwise direction, until all load is assigned. After all traffic class assignments are made, in each site, we determine the sizes of the partitions that host each traffic class. To compute the partition sizes, we use a known gradient descent algorithm [62] that minimizes the midgress of each site. The total midgress achieved by `baseline fit-part` is then the sum of the midgress across all sites.

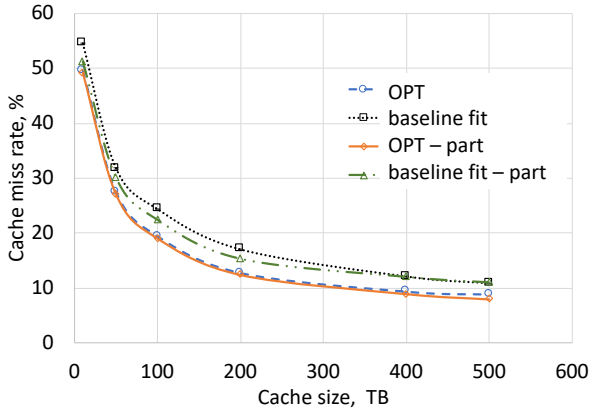

Figure 7: MRC of OPT and baseline fit on shared and partitioned caches.

Experimental evaluation. We evaluate `baseline fit-part` and OPT-part at the metro-level using production traces described in Section 4 and at different cache sizes per cluster of 1 TB, 5 TB, 10 TB, 20 TB, 40 TB and 50 TB. We report the average cache miss rate of 100 runs. The 95% confidence intervals of the expected cache miss rates have a margin of error of less than 0.4%.

As shown in Figure 7 we find that OPT-part reduces midgress when compared to `baseline fit-part` by more than 14%, on average across the different cache sizes. Thus midgress-aware traffic provisioning can significantly reduce the midgress even for partitioned caches. As comparison, in Figure 7, we also plot OPT and `baseline fit` that we described for shared caches in Sections 2 and 3. Interestingly, we find that the cache miss rate of OPT-part is only 0.49% less than that of OPT, on average across the different cache sizes. Hence, while OPT-part has the lowest cache miss rate,

OPT has nearly the same miss rate without the additional overhead of cache partitioning.

5.3.1 Implementing cache partitioning for traffic provisioning in production settings

In the previous section, we have seen that cache partitioning can further reduce the midgress of a metro area since each traffic class occupies a separate partition and traffic classes assigned to the same site could have different eviction ages. But it is not implemented in practice for traffic provisioning due to the following reasons.

1) In large CDNs, many different traffic classes must share a cache. Further, the mix of traffic classes sharing a cache and their respective loads change frequently over time at the whim of the load balancer. To obtain the benefits of cache partitioning, the partitions need to be constantly resized by shrinking cache space for certain traffic classes and expanding the cache space for others. Such resizing is resource intensive. Further, constantly resizing dynamic partitions may not lead to lower midgress, especially during transitions between cache sizes. While partitioning the cache statically is easier to implement, static partitions do not adjust to changes in the traffic mix, leading to sub-optimal performance. On the other hand, an unpartitioned shared cache dynamically adjusts the cache space occupied by different traffic classes based on the load and the traffic characteristics, without the need for complex (re)partitioning operations.

2) From the previous section, we see that traffic provisioning in a shared unpartitioned cache provides nearly the same midgress as a partitioned cache. Thus, there is little incentive to redesign the traffic provisioning system to work with partitioned caches and incurring the additional software complexity and resource overhead.

For the reasons outlined above, the shared unpartitioned cache studied in our work is the implementation of choice for many major CDNs, including Akamai.

6 Related work

Traffic provisioning in CDNs has been studied in the context of load balancing. However, the load balancing algorithms focus on ensuring that servers are not overloaded and do not explicitly minimize midgress. Likewise, minimizing cache misses through better cache management policies has a rich literature. However, we view cache management as complementary to midgress-aware traffic provisioning. We review relevant existing literature in these areas.

Load balancing. Request redirection schemes at the network layer, based on DNS [8, 32], and at the application layer, based on URL rewriting or HTTP redirection [48], have been proposed to load balance traffic across multiple servers. Dynamic load balancing algorithms [10, 11, 71] continuously measure the load on different servers and load balance end-user requests to improve performance. Consistent-hashing and randomized load balancing algorithms [41, 42, 54, 55] have also been proposed to load balance end-user requests in content delivery systems. Extensions to traditional load balancing, that minimize the energy consumption of CDNs [50] have also been proposed in the literature. Much of the work above are in the context of routing user requests in real-time to servers. But, they can be adapted to our context of performing (offline) traffic provisioning, a step that precedes request routing in a production CDN. *However, there is no prior work on explicitly minimizing midgress.*

Cache management. There has been a significant amount of research on cache management policies to minimize cache miss rates [2, 3, 6, 17–19, 22, 26–31, 33, 37, 38, 45, 49, 52, 57, 58, 60, 64, 68, 70]. Some proposed caching policies include Adapt-Size [4], Cliffhanger [15], SLRU [40], TLRU [23], S4LRU [34], CFLRU [59], ARC [53], LRU-S [65], LRU-K [56], and GDS [7]. Dynamically partitioning the cache to reduce miss rates has also been explored [1, 5, 9, 14, 16, 21, 25, 36, 43, 44, 46, 62, 67, 69]. However, production CDNs do not employ dynamically-partitioned caches since it introduces significant performance and operational overheads. We view work on cache management as a complementary technique to traffic provisioning, both with the goal of midgress reduction.

Recent work on footprint descriptors [66] is focused on efficient techniques for evaluating the miss rates of traffic mixes. We use footprint descriptors to quickly compute the midgress of a traffic class assignment, as well as to efficiently compute the MRC and eviction age function of traffic classes. However, the work on footprint descriptors does not minimize midgress.

7 Conclusion

We propose midgress-aware traffic provisioning that explicitly minimizes the midgress traffic of a CDN, while ensuring that no server or cluster is overloaded. Using extensive traces for 25 traffic classes from Akamai's CDN, we show that the midgress of a metro can be reduced by 18.37% when compared to a midgress-unaware baseline. We propose a midgress-aware heuristic, `local search`, that provisions traffic classes to achieve a midgress reduction that is within 1.1% of the optimum, and is very fast and well suited for production settings. We also show that using our traffic provisioning algorithms at the cluster level results in significant reductions in midgress. Given that a large CDN can have midgress of over 10 Tbps, even a small reduction in midgress can result in millions of dollars of savings per year. Our work provides a strong case for implementing midgress-aware provisioning in CDNs.

8 Acknowledgments

We thank our reviewers and our shepherd Kiran Kumar Muniswamy Reddy for their great feedback. This research was supported in part by NSF grant CNS-1763617.

References

[1] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. Cloudcache: On-demand flash cache management for cloud computing. In *FAST*, pages 355–369, 2016.

[2] Daniel S. Berger, Philipp Gland, Sahil Singla, and Florin Ciucu. Exact analysis of TTL cache networks. *Perform. Eval.*, 79:2 – 23, 2014. Special Issue: Performance 2014.

[3] Daniel S Berger, Sebastian Henningsen, Florin Ciucu, and Jens B Schmitt. Maximizing cache hit ratios by variance reduction. *ACM SIGMETRICS Performance Evaluation Review*, 43(2):57–59, 2015.

[4] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *NSDI*, pages 483–498, 2017.

[5] Sem Borst, Varun Gupta, and Anwar Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. Citeseer, 2010.

[6] PJ Burville and JFC Kingman. On a model for storage and search. *Journal of Applied Probability*, pages 697–701, 1973.

[7] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *USENIX symposium on Internet technologies and systems*, volume 12, pages 193–206, 1997.

[8] Valeria Cardellini, Michele Colajanni, and Philip S. Yu. Request redirection algorithms for distributed web systems. *IEEE transactions on parallel and distributed systems*, 14(4):355–368, 2003.

[9] Damiano Carra and Pietro Michiardi. Memory partitioning and management in memcached. *IEEE Transactions on Services Computing*, 2016.

[10] Robert L Carter and Mark E Crovella. Server selection using dynamic path characterization in wide-area networks. In *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 3, pages 1014–1021. IEEE, 1997.

[11] Chung-Min Chen, Yibei Ling, Marcus Pang, Wai Chen, Shengwei Cai, Yoshihisa Suwa, and Onur Altintas. Scalable request routing with next-neighbor load sharing in multi-server environments. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, volume 1, pages 441–446. IEEE, 2005.

[12] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 167–181. ACM, 2015.

[13] Ludmila Cherkasova. *Improving WWW proxies performance with greedy-dual-size-frequency caching policy*. Hewlett-Packard Laboratories, 1998.

[14] Weibo Chu, Mostafa Dehghan, John CS Lui, Don Towsley, and Zhi-Li Zhang. Joint cache resource allocation and request routing for in-network caching services. *Computer Networks*, 131:1–14, 2018.

[15] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. Cliffhanger: Scaling performance cliffs in web memory caches. In *USENIX NSDI*, pages 379–392, 2016.

[16] Asaf Cidon, Daniel Rushton, Stephen M Rumble, and Ryan Stutsman. Memshare: a dynamic multi-tenant key-value cache. In *Usenix ATC*, 2017.

[17] Edward G. Coffman and Predrag Jelenković. Performance of the move-to-front algorithm with Markov-modulated request sequences. *Operations Research Letters*, 25:109–118, 1999.

[18] Edward Grady Coffman and Peter J Denning. *Operating systems theory*. Prentice-Hall, 1973.

[19] Asit Dan and Don Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *ACM SIGMETRICS*, pages 143–152, 1990.

[20] Mostafa Dehghan, Weibo Chu, Philippe Nain, Don Towsley, and Zhi-Li Zhang. Sharing cache resources among content providers: A utility-based approach. *IEEE/ACM Transactions on Networking (TON)*, 27(2):477–490, 2019.

[21] Mostafa Dehghan, Laurent Massoulie, Don Towsley, Daniel Menasche, and Yong Chiang Tay. A utility optimization approach to network cache design. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[22] Robert P Dobrow and James Allen Fill. The move-to-front rule for self-organizing lists with Markov dependent requests. In *Discrete Probability and Algorithms*, pages 57–80. Springer, 1995.

[23] Gil Einziger and Roy Friedman. Tinylfu: A highly efficient cache admission policy. In *IEE Euromicro PDP*, pages 146–153, 2014.

[24] Ronald Fagin. Asymptotic miss ratios over independent references. *Journal of Computer and System Sciences*, 14(2):222–250, 1977.

[25] Michal Feldman and John Chuang. Service differentiation in web caching and content distribution. In *Proceedings of the IASTED International Conference on Communications and Computer Networks*, 2002.

[26] James Allen Fill and Lars Holst. On the distribution of search cost for the move-to-front rule. *Random Structures & Algorithms*, 8:179–186, 1996.

[27] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39:207–229, 1992.

[28] Christine Fricker, Philippe Robert, and James Roberts. A versatile and accurate approximation for LRU cache performance. In *ITC*, page 8, 2012.

[29] Massimo Gallo, Bruno Kauffmann, Luca Muscariello, Alain Simonian, and Christian Tanguy. Performance evaluation of the random replacement policy for networks of caches. In *ACM SIGMETRICS/ PERFORMANCE*, pages 395–396, 2012.

[30] Nicolas Gast and Benny Van Houdt. Transient and steady-state regime of a family of list-based cache replacement algorithms. In *ACM SIGMETRICS*, pages 123–136, 2015.

[31] Erol Gelenbe. A unified approach to the evaluation of a class of replacement algorithms. *IEEE Transactions on Computers*, 100:611–618, 1973.

[32] Michel Goemans. Load balancing in content deliverynetworks. *MA Annual Program Year Workshop:Network Management and Design*, April 2003.

[33] WJ Hendricks. The stationary distribution of an interesting Markov chain. *Journal of Applied Probability*, pages 231–233, 1972.

[34] Qi Huang, Ken Birman, Robbert van Renesse, Wyatt Lloyd, Sanjeev Kumar, and Harry C Li. An analysis of Facebook photo caching. In *ACM SOSP*, pages 167–181, 2013.

[35] Cisco Visual Networking Index. The zettabyte era: Trends and analysis. June 2017.

[36] Stratis Ioannidis and Edmund Yeh. Jointly optimal routing and caching for arbitrary network topologies. *IEEE Journal on Selected Areas in Communications*, 2018.

[37] Predrag R Jelenković. Asymptotic approximation of the move-to-front search cost distribution and least-recently used caching fault probabilities. *The Annals of Applied Probability*, 9:430–464, 1999.

[38] Predrag R Jelenković and Ana Radovanović. Least-recently-used caching with dependent requests. *Theoretical computer science*, 326:293–327, 2004.

[39] Poul-Henning Kamp. Varnish LRU architecture, June 2007. Available at https://www.varnish-cache.org/trac/wiki/ArchitectureLRU, accessed 09/12/16.

[40] Ramakrishna Karedla, J Spencer Love, and Bradley G Wherry. Caching strategies to improve disk system performance. *Computer*, (3):38–46, 1994.

[41] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[42] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. *Computer Networks*, 31(11):1203–1213, 1999.

[43] Terence Kelly, Yee Man Chan, Sugih Jamin, and Jeffrey MacKie-Mason. Biased replacement policies for web caches: Differential quality-of-service and aggregate user value. 1999.

[44] Seongbeom Kim, Dhruba Chandra, and Yan Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 111–122. IEEE Computer Society, 2004.

[45] W. Frank King. Analysis of demand paging algorithms. In *IFIP Congress (1)*, pages 485–490, 1971.

[46] Bong-Jun Ko, Kang-Won Lee, Khalil Amiri, and Seraphin Calo. Scalable service differentiation in a shared storage cache. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 184–193. IEEE, 2003.

[47] Bruce M Maggs and Ramesh K Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

[48] Sabato Manfredi, Francesco Oliviero, and Simon Pietro Romano. A distributed control law for load balancing

in content delivery networks. *IEEE/ACM Transactions on Networking (TON)*, 21(1):55–68, 2013.

[49] Valentina Martina, Michele Garetto, and Emilio Leonardi. A unified approach to the performance analysis of caching systems. In *IEEE INFOCOM*, 2014.

[50] Vimal Mathew, Ramesh K Sitaraman, and Prashant Shenoy. Energy-aware load balancing in content delivery networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 954–962. IEEE, 2012.

[51] Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[52] John McCabe. On serial files with relocatable records. *Operations Research*, 13:609–618, 1965.

[53] Nimrod Megiddo and Dharmendra S Modha. ARC: A self-tuning, low overhead replacement cache. In *USENIX FAST*, volume 3, pages 115–130, 2003.

[54] Vahab Mirrokni, Mikkel Thorup, and Morteza Zadimoghaddam. Consistent hashing with bounded loads. *arXiv preprint arXiv:1608.01350*, 2016.

[55] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[56] Elizabeth J O'Neil, Patrick E O'Neil, and Gerhard Weikum. The LRU-K page replacement algorithm for database disk buffering. *ACM SIGMOD*, 22(2):297–306, 1993.

[57] Elizabeth J O'Neil, Patrick E O'Neil, and Gerhard Weikum. An optimality proof of the LRU-K page replacement algorithm. *JACM*, 46:92–112, 1999.

[58] Antonis Panagakis, Athanasios Vaios, and Ioannis Stavrakakis. Approximate analysis of LRU in the case of short term correlations. *Computer Networks*, 52:1142–1152, 2008.

[59] Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jinsoo Kim, and Joonwon Lee. CFLRU: a replacement algorithm for flash memory. In *ACM/IEEE CASES*, pages 234–241, 2006.

[60] Konstantinos Psounis, An Zhu, Balaji Prabhakar, and Rajeev Motwani. Modeling correlations in web traces and implications for designing replacement policies. *Computer Networks*, 45:379–398, 2004.

[61] Guocong Quan, Jian Tan, Atilla Eryilmaz, and Ness Shroff. A new flexible multi-flow lru cache management paradigm for minimizing misses. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):39, 2019.

[62] Moinuddin K Qureshi and Yale N Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 423–432. IEEE, 2006.

[63] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

[64] Eliane R Rodrigues. The performance of the move-to-front scheme under some particular forms of Markov requests. *Journal of applied probability*, pages 1089–1102, 1995.

[65] David Starobinski and David Tse. Probabilistic methods for web caching. *Perform. Eval.*, 46:125–137, 2001.

[66] Aditya Sundarrajan, Mingdong Feng, Mangesh Kasbekar, and Ramesh Sitaraman. Footprint descriptors: Theory and practice of cache provisioning in a global cdn. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 55–67. ACM, 2017.

[67] Dominique Thiébaut, Harold S. Stone, and Joel L Wolf. Improving disk cache hit-ratios through cache partitioning. *IEEE Transactions on Computers*, 41(6):665–676, 1992.

[68] Naoki Tsukada, Ryo Hirade, and Naoto Miyoshi. Fluid limit analysis of FIFO and RR caching for independent reference model. *Perform. Eval.*, 69:403–412, September 2012.

[69] Ying Ye, Richard West, Zhuoqun Cheng, and Ye Li. Coloris: a dynamic cache partitioning system using page coloring. In *Parallel Architecture and Compilation Techniques (PACT), 2014 23rd International Conference on*, pages 381–392. IEEE, 2014.

[70] Neal E Young. Online paging against adversarially biased random inputs. *Journal of Algorithms*, 37:218–235, 2000.

[71] Zeng Zeng and Bharadwaj Veeravalli. Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks. *IEEE Transactions on Computers*, 55(11):1410–1422, 2006.