Darwin: Flexible Learning-based CDN Caching

Jiayi Chen[1], Nihal Sharma[1], Tarannum Khan[1], Shu Liu[2], Brian Chang[1], Aditya Akella[1], Sanjay Shakkottai[1], Ramesh K. Sitaraman[3]
[1]The University of Texas at Austin, [2]UC Berkeley, [3]UMass Amherst & Akamai Tech

ABSTRACT

Cache management is critical for Content Delivery Networks (CDNs), impacting their performance and operational costs. Most production CDNs apply static, hand-tuned caching policy parameters at cache servers, such as admission frequency or size thresholds for the Hot Object Caches (HOC) of their system. However, these static policies fall short when a server is faced with unpredictable traffic pattern changes, even when policies employ multiple control parameters/knobs. Recent approaches have proposed learning-based solutions to dynamically adjust policy parameters, but they are limited in action space, caching objectives, or impose high overhead. We propose Darwin, a CDN cache management system that is robust to traffic pattern changes and can flexibly optimize different caching objectives with unrestricted action spaces. Darwin employs a three-stage pipeline involving traffic pattern feature collection, unsupervised clustering for classification, and neural bandit expert selection to choose the optimal caching policy. Through extensive simulations, experiments using an Apache Traffic Server (ATS)-based prototype, and theoretical analysis, we show that Darwin achieves significant performance gain w.r.t. different objectives such as maximizing object hit rates and minimizing disk writes, while simultaneously adapting to traffic pattern shifts. Darwin imposes negligible overhead and achieves high throughput compared to the state-of-the-art.

CCS CONCEPTS

• **Computer systems organization** → **Processors and memory architectures**; • **Computing methodologies** → **Machine learning approaches**;

KEYWORDS

Content Delivery Networks, Cache Management, Machine Learning

ACM Reference Format:
Jiayi Chen, Nihal Sharma, Tarannum Khan, Shu Liu, Brian Chang, Aditya Akella, Sanjay Shakkottai, Ramesh K. Sitaraman. 2023. Darwin: Flexible Learning-based CDN Caching. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3603269.3604863

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3604863

1 INTRODUCTION

Content Delivery Networks (CDNs) [14, 31] serve most of the Internet traffic today. They enhance the user experience by deploying a large number of edge servers that can cache content close to the clients (i.e., users). A large production CDN, such as Akamai's, may deploy more than 350,000 servers across 1400+ networks in 100+ countries [3].

A CDN server has a hierarchical structure with the Hot Object Cache (HOC) comprising in-memory storage for fast access, and the Disk Cache (DC) which has much more storage, but with slower access. An incoming request to the CDN is served from the HOC if the object exists there, else it is served from the DC. If the object is unavailable at the DC as well, it is retrieved from the origin site of the content provider. Cache management policies that determine which objects to hold versus evict in the HOC or DC play a key role in improving user-perceived latencies.

CDNs employ multi-level load balancing to determine how CDN servers cache and serve content (Section 2.1). These load balancing policies may conspire to impose significant traffic pattern changes at a given CDN server; over time, a server may see requests for objects from different "traffic classes" (e.g., images or video segments or web pages or software downloads; Section 3), causing significant changes in key request attributes such as the request frequency, object size, and recency distributions [10, 42].

Unfortunately, this variability has a significant impact on the effectiveness of CDN caching and renders widely-used cache management policies ineffective [10, 39]. This makes it challenging to design cache management policies that adapt to observed traffic patterns.

We consider the problem of cache management policy design in the context of *admission policies* that determine when/which objects are stored in the limited-capacity HOC. We find that an admission policy cannot simply consider a single knob, e.g., a threshold f on the frequency of requests, to determine whether to cache in the HOC. Furthermore, multi-knob static policies (for example, caching objects of size $\leq s$ that have been requested $\geq f$ times) also fall short (Section 3).

Learning-based approaches that avoid the pitfall of static choices have been considered for cache management in general [2, 39, 44] and HOC admission in particular [10]. However, prior learned approaches suffer from one or more key drawbacks rendering them impractical (Section 3): they consider a single knob and cannot be easily extended to several; their algorithms target a specific objective, such as object hit rate, and cannot be easily applied to others that combine hit rates with operational costs; and, their approaches induce high overhead due to having to look up complex inference models for every request. Furthermore, many of the approaches lack sound theoretical backing.

We develop Darwin, a practical and provably effective approach to adaptive learning-based HOC admission that addresses the above issues. At its core, Darwin is based on a novel approach to neural-aided bandit selection. Using this approach, Darwin makes an *online selection of the "best-expert" HOC admission policy*, where experts are defined by thresholds on the aforementioned knobs. The number of experts can be large depending on the number of knobs and the values they can take, and evaluating them to make an informed runtime choice can be prohibitive. Darwin scales the problem down by using two ideas that work offline: (1) performing *unsupervised clustering of workloads* and associating with each cluster a small subset of experts offering good performance with respect to the chosen objective; and (2) training *cross-expert prediction neural nets* that eliminate the need for direct evaluation of expert performance by enabling prediction of the runtime performance of an expert based on a different expert that is currently running. Our online bandit selection algorithm uses the above steps as sources of *side information* – it maps incoming traffic to a cluster and uses the prediction networks to select the best expert from among the ones corresponding to the cluster. We prove that with high probability our algorithm identifies the best expert in a finite amount of time that is constant with respect to the number of experts.

A key advantage of Darwin over existing caching approaches is that it can be used to optimize for different types of metrics. In particular, most prior literature on caching focuses on optimizing *hardware-independent* metrics like hit rate. This is because the prior algorithms use a hardware-independent model for a cache; e.g., Belady's algorithm [8] has an optimal hit rate independent of the hardware it runs on. As a result, prior approaches do not apply to *hardware-dependent* (resource-related) metrics such as writes to storage that are very important in practice. In contrast, an appealing aspect of Darwin's approach is that it is based on actually running experts on the given hardware. Thus, it can optimize *both* types of metrics. For the same traffic, Darwin can pick different experts for different hardware configurations given the necessary offline training.

Overall, Darwin's design allows it to be highly customizable. CDN operators can use the same framework for different traffic features that can be collected, different objectives that combine cache performance with costs, and different knob choices that are viable in a given deployment.

We build a prototype of Darwin that uses experts parameterized by two knobs $<f, s>$, atop the Apache Traffic Server [18]. We conduct an extensive evaluation using both simulations and real deployment, and both synthetic and real traces derived from a production CDN server. Some highlights from our findings include: (1) Darwin improves the object hit rate (OHR) by 3-43% compared to static baselines and state-of-the-art HOC admission policies. (2) We show that Darwin can be used to improve other metrics, e.g., a linear combination of OHR and disk writes improves by up to 97% and byte miss ratio by 11% compared to static approaches. (3) Darwin imposes minimal CPU overhead and achieves a throughput of up to 10.4Gbps, outperforming static experts due to its higher hit rates.

2 BACKGROUND

We now describe CDNs, the cache management system utilized by CDN servers (alternately, CDN cache), and the main metrics used to evaluate a CDN cache.

2.1 Content Delivery Networks

The first point-of-contact for any client request is the closest CDN server, which probes its local cache for the object. On a cache hit (when the object is found in cache), the server retrieves this content from memory and delivers it to the client, thus minimizing download times which leads to better performance. In contrast, on a cache miss, the server must source this object from the origin server over the Wide Area Network (WAN) before it can relay it to the client. This additional effort translates to higher latency at the client, thus degrading performance. Additionally, on a cache miss, the server also decides whether or not to place this retrieved object in its cache to serve future requests. Besides worse performance, a cache miss also results in extra bandwidth usage due to "midgress" traffic between the CDN server and the origin [42, 50]. Hence, increasing cache hits (and decreasing cache misses) is the holy grail for CDN operators.

A traffic class [41, 42] is a set of domain names with a particular content type such as images or text from a content provider(s) (such as a social media site) with similar access characteristics. A CDN server serves requests from highly diverse sets of traffic classes. Due to the differences in the content they represent, these classes also display a large variety in access frequency and size statistics. Two levels of load balancers [12, 27] work in tandem to choose a specific CDN server to serve each request. The first of these, the global load balancer, performs inter-cluster distribution, while a second local load balancer dispenses traffic between servers within a cluster. The goal of CDN load balancing is to efficiently route client requests to servers that are most likely to contain the requested content in their local cache. CDN load balancing is often performed via DNS with TTLs that are set to be small [36]. This allows the load balancer to react quickly to changes in traffic demand, server state, and network state. Akamai, for example, employs a TTL of about 20s to help the load balancer modulate traffic mixes to its servers. As access patterns, network conditions, and server states change rapidly, these load balancers continually adjust the traffic class mixes of each CDN server to meet availability, performance, and capacity constraints. For instance, a CDN server that is serving mostly small objects from a Web traffic class may start to serve larger objects from a software download class when an important iOS update is released. *The volume and mix of traffic classes assigned to a CDN server can change rapidly requiring a flexible cache management system that adapts to the change without a deterioration in cost and performance, designing such as system is the main focus of our work.*

2.2 Cache Management System

A CDN server has a *hierarchical* structure with a Hot Object Cache (HOC) and a Disk Cache (DC) as shown in Figure 1 [10]. HOC is a fast but small first-level cache that resides in main memory and is used to store frequently requested objects so that the objects can be accessed quickly. In contrast, the DC is high-capacity, with slower access time.

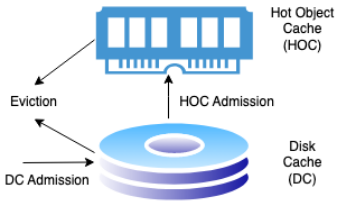


Figure 1: A two-level CDN cache serves requests with Hot Object Cache (HOC) and Disk Cache (DC). It can deploy various combinations of admission/eviction policies to optimize cache usage.

The CDN server's cache management system decides which objects to *admit* in each of its caches and which to *evict* when the cache is full as shown in Figure 1. Upon a cache miss (the object is not present in the HOC or DC and is fetched from the origin), the cache management system determines whether or not the object can be admitted into the DC. With a cache hit (object is already present in the DC), the object may also be admitted (i.e., promoted) into the HOC. The decision to admit an object into either cache is made in accordance with the *admission policy* of that cache. If the DC or the HOC is full, one or more objects will need to be evicted to make room for the newly-admitted object. The decision of which object(s) to evict from either cache is determined by the *eviction policy* of the respective cache.

Caching Policies. There is a vast literature on eviction policies [2, 7, 13, 32, 44]. These generally rely on object metrics such as recency and frequency of access, size, and cost of a cache miss [21]. The most commonly deployed policies are based on the Least Recently Used (LRU) strategy (evict objects not requested for the longest interval).

In contrast, admission policies have received much less attention. Nearly 70% of the unique objects accessed from a CDN cache are "one-hit wonders", i.e., objects that are only accessed once [27]. Admitting one-hit wonders into the DC is a waste of disk resources since they are never accessed again. Thus, a common admission policy is to only admit objects into the DC when it is requested for a second time by recording (but not admitting) the first request in a bloom filter [27]. Unlike the DC, the HOC has limited capacity. So, attributes such as frequency f [17, 40] and size s [10] are often also considered for HOC admissions. For example, the HOC may admit an object if it has a size smaller than a threshold s or when the requested object has been accessed with a frequency more than a threshold f.

CDN Caching Objectives: A cache management system is designed to optimize several objective metrics, each metric significantly impacting either the performance (e.g., latency) or cost (e.g., bandwidth cost).

Hit-rate metrics: A metric that directly measures caching performance is the *object hit rate* (OHR, or simply, hit rate), i.e., the ratio of the number of cache hits to the total number of requests served. Each cache hit saves bandwidth and time to fetch the object over the WAN. High OHRs improve object retrieval latencies, and consequently the client-perceived response times. A related metric of *byte miss rate* (BMR), i.e., the ratio of bytes served during cache misses to the total number of bytes served, is also tracked and minimized. The midgress traffic from the CDN servers to the origin due to cache misses is directly proportional to BMR. The bandwidth cost of *midgress traffic* is a significant portion of the operating expenditures (OPEX) of a CDN [42], requiring the minimization of BMR.

Resource-related metrics: In addition to the hit-rate related metrics, cache management systems also optimize resource-related metrics such as disk operations needed to serve the content. Primary among them is *SSD writes* that impact both the SSD utilization and lifetimes. Notably, excessive SSD writes can cause the disk to reach its write endurance limit, requiring additional capital expenditures (CAPEX) for replacing the disks or the servers [35, 50, 52].

A variety of heuristic policies have been explored for cache management systems that optimize one or the other of the above hit-rate metrics [5, 17, 20, 47]. In contrast, limited attention has been paid to policies for resource-related metrics (see, e.g., [28]) which are also important in practice. Irrespective, these heuristics operate one or more decision knobs each taking a range of values. Together, they lead to a huge space of tunable parameters for CDN cache management.

3 MOTIVATION

The problem of eviction is well-studied, with classical results dating back decades [7, 13, 32] to more recent work on data-driven approaches [2, 44]. Our work, in contrast, focuses on admission policies, which are less well-studied. As recent studies have observed [10], HOC admission policies play a crucial role in overall CDN caching performance and client-perceived response times.

In this section, we argue that admission policies need multiple control knobs – simply admitting based on frequency or size threshold, for example, is insufficient. More importantly, we show that static admission policies are rather sub-optimal – *even when considering multiple knobs*. This makes a case for learning-based admission policies, but we find that current approaches suffer from fundamental drawbacks.

3.1 Drawbacks of Static Policies

As mentioned in Sec. 2.1, the load balancer of a CDN can rapidly vary both the traffic volumes and traffic classes that are assigned to a CDN server. Since different traffic classes have widely varying characteristics (e.g. different object size and popularity distributions and patterns of access), a CDN server must alter its caching strategy in a dynamic fashion to adapt to changes in the traffic mix.

In production settings, CDN HOC admission typically relies on hand-tuned parameters, such as predetermined frequency (f) and size (s) thresholds. Unfortunately, there is no one-size-fits-all policy; different traffic mixtures lead to different settings of the control knobs (controlling either one or multiple parameters) being optimal.

Example with a real trace. We simulate a CDN cache server's behavior using real traces collected from a production CDN server. We simulate a 100MB HOC (with a bloom filter for one-hit wonders) and a 10GB DC on these traces, using LRU as our eviction algorithm. Figure 2a and 2b show the HOC OHR under various static f and s thresholds for two *randomly-picked* time windows, each with 2M requests, in a CDN production subtrace from two distinct media traffic classes in Europe.

From Figure 2a, we see that simply controlling one knob – e.g., fixing f and ignoring s – is not sufficient.[1] For a given choice of f (likewise s) a specific non-trivial choice of s (f) is needed to optimize OHR. While our analysis here considers just two knobs, HOC policies could use additional knobs (e.g., recency and frequency per-unit size) that can play a crucial role in caching alongside f and s. We study a third dimension (recency) in Section 6.

Furthermore, we see that sticking to the best parameters for window 1 (f=3, s=7MB) causes the window 2 requests to perform 1.19% worse than optimal (f=1, s=7MB), while keeping window 2's best parameters (f=1, s=7MB) degrades window 1's HOC hit rate by 7.83%.[2]

We posit that an underlying reason for these observations is a shift in workload – either in the mixes between different traffic classes or in the size/frequency distribution of objects. To understand the impact of and dependence on traffic composition, we consider an extreme case where objects are from just one traffic class as seen at the CDN server. Specifically, we examine the Image (Figure 2c) and Download traffic (Figure 2d, 2e) class subsets of a production server trace.

The Image class has many requests for infrequently accessed objects and 71.9% of the requests are for objects whose sizes are smaller than 20KB. The best HOC admission parameters for the Image class are f=5 and s=20KB, as shown in Figure 2c. Larger frequency thresholds prevent those two-hit/three-hit/four-hit wonders from entering the HOC, and therefore better utilize the limited memory space with the more popular objects. On the other hand, the size threshold of 20KB gives most of the objects a chance to reside in the HOC, while preventing the few large objects from taking up the space that can serve more objects.

However, we can see from Figure 2d that for Download requests, the previous hand-tuned admission thresholds (f=5, s=20KB) become suboptimal. Choosing another parameter set (f=1, s=5MB) for the Download class can reach a 71.39% higher HOC hit rate. This is because the Download class objects are more popular. Increasing the frequency threshold doesn't have a large effect on which objects get admitted to HOC as these objects all have more than 7 requests. A larger frequency threshold slightly hurts the HOC OHR performance because the HOC admission of each object takes more requests. On the other hand, the subset of common media objects have much larger object sizes and only 21.5% of the requests are for objects below 50KB. They need a larger size threshold to keep the most common but reasonably-sized objects in the HOC cache.

3.2 Learning the Admission Decisions

Monitoring traffic properties and tuning policies manually is difficult. One might ask if the optimal admission decisions are learnable. Indeed, CDN caches have key properties that are beneficial for learning: (1) Diverse CDN servers produce large amounts of logs every day, creating a large and diverse dataset that can be used to learn from. (2) It is easy to obtain the cache performance representation (footprint descriptor), even from completely anonymized logs [41] of CDN requests – this is strongly correlated with the traffic's cache performance. (3) The best caching behaviors are deterministic for a given request sequence, and therefore learnable by considering prior traces. (4) CDN cache servers in general are not CPU-bound[39], and can leverage available compute toward learning-based decisions.

Next, we describe the requirements for caching with learned HOC admission and describe where prior art falls short.

Name	Year	Many Knobs	Diverse Goals	Low Overhead	Theoretical Guarantees
Darwin	2023	✓	✓	✓	✓
LHR [49]	2021	✓	✗	✗	✗
RL-Cache [22]	2019	✓	✓	✗	✗
AdaptSize [10]	2017	✗	✗	✓	✗
Hill Climbing [10]	2017	✓	✓	✗	✗
Percentile [10]	2017	✓	✓	✓	✗

Table 1: Learned cache admission schemes (Percentile and Hill Climbing discussed in Section 6)

3.2.1 *Requirements, and Issues with Prior Schemes.*

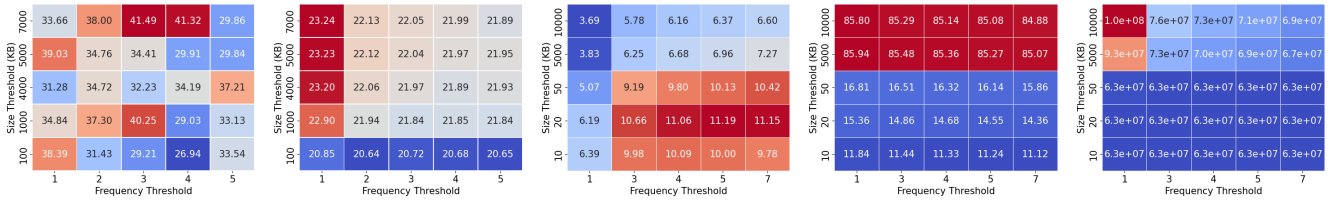
Adaptation. The primary requirement of a learned admission scheme is that as traffic mixtures of different traffic classes expected at a CDN server change, the cache management policy should be able to *(R1) adapt to best suit the current traffic*. In particular, the policy should offer performance very close (e.g., within, say 1% in terms of the OHR) to the "hindsight optimal" policy, since directly matching the performance of an oracle is impossible.

Multiple decision knobs. Decision knobs such as f and s are the parameters that a learning-based approach uses in its decision-making and define the approach's action spaces. Restricting the knobs restrains the possible action space that the admission policies can work with. AdaptSize, for example, learns the probabilistic *size threshold* (s) for HOC admission. It develops a Markov chain model that estimates OHR as a function of the object size threshold. However, with the same input features and models, AdaptSize cannot extend its model to other admission heuristics that also involve other knobs, e.g., f and s. This can be problematic: e.g., when a subset of popular objects is mixed with a cache scan [33], AdaptSize's size-based policy is suboptimal as it doesn't consider object frequency and admits objects with low popularity. Similar issues arise with admission rules based on frequency alone [17, 27]. Thus, *(R2) learning-based approaches should accommodate multiple decision knobs to cover more advanced admission policies.*

CDN optimization goal. This refers to the metric that the learning approach optimizes for. Cache servers can have diverse goals. For example, a server with SSD as the disk cache may want to optimize disk writes while maintaining high OHR. Most approaches today employ algorithms that target a specific goal, which is often hardware-independent (e.g., OHR). This can cause highly suboptimal behavior with respect to other goals; for example, as shown in Figure 2d and 2e, the static frequency and size thresholds that achieve the highest OHR for the Download trace offers the second highest SSD writes. The underlying algorithms cannot be easily extended to other goals, especially hardware-dependent ones. For example, we cannot easily adapt AdaptSize's approach, which relates OHR to a probabilistic size admission threshold model, to optimize SSD writes due to the complexity of modeling hardware behavior. Other cache management approaches, such as those based

[1] Note that because of the use of the bloom filter, a particular value of f implies that an object is let into the HOC upon the $(1+f)^{st}$ request.

[2] Compared to these randomly chosen windows, in our end-to-end evaluation in Section 6 we observe much more significant differences between static expert choices and experts chosen in an adaptive manner.



(a) Production (Win1) OHR (b) Production (Win2) OHR (c) Image OHR (d) Download OHR (e) Download Disk Write

Figure 2: (a) (b): HOC OHR results for two windows in a CDN production trace with different frequency and size thresholds. (c) (d) (e): The optimal parameters change with different traces and evaluation metrics. (c) HOC OHR for the Image trace is optimal with (f=5, s=20KB). (d) HOC OHR for the Download trace is optimal with (f=1, s=5MB). (e) Disk write for the Download trace is optimal with (f=1, s=10MB).

on Hill Climbing [10] face similar issues. Ideally, *(R3) the cache management policy should be easily customized to all types of different objectives*, avoiding baking the objective into the design.

Overhead. A learning-based cache management approach should *(R4) impose low overhead on the system overall.* Some approaches, e.g., RL-Cache [22] and LHR [49] learn the features of multiple objects using which they predict the current object's decision (object-based learning). However, such prediction-per-request is significantly more expensive than approaches that, by design, invoke learning for every time interval or once every several requests. Additionally, learning approaches like hill climbing [10] require simultaneous runs of shadow caches to report online statistics of the policies in comparison, which induces high memory overhead.

Table 1 compares learning-based mechanisms explored in prior work that can be applied to tune CDN cache admission policies. All proposals attempt to meet R1, although to different levels of effectiveness (Section 6), but fail on one or more of the other requirements. None of the approaches have theoretical backing either.

4 DARWIN DESIGN

We present Darwin in the context of admission policies that maximize the HOC hit rate in a CDN server. We show how it extends to other practical objectives in Section 6. There are several possibilities for applying learning to this problem. We initially considered training a learning algorithm to make per-object admission predictions, but this has a high overhead violating R4. Furthermore, speaking to CDN operators highlighted hesitation in deploying this approach due to the black-box nature of predictions and difficulty with interpreting decisions.

We then considered a more practical approach that maps features of arriving traffic *directly to the available knobs* of a HOC admission policy (e.g., f or s or jointly predict both). This ensures better interpretability, but we found that its OHR performance is poor (Section 6) mainly because there was no way to control the inherent error in the approach's parameter prediction. Also, predicting the best joint parameter choice is challenging when the number of knobs is large.

Darwin's approach is to use learning to "test" and select among a set of "known good experts". This preserves interpretability while avoiding the pitfalls of direct prediction. In Darwin, each expert is characterized by a tuple (f, s) of frequency and size thresholds, and promotes to HOC all objects that occur more than f times and request objects of size lesser than s. Darwin learns to associate traffic patterns in the arriving workload to the best-performing expert (one with the largest hit rate) in the given set. Darwin can be trivially extended to include other knobs.

Our approach is summarized in Figure 3 and is split into two steps: offline training, and online selection.

1. Offline training (Section 4.1, Appendix A.1) :

1a. Offline clustering and expert set association: We collect historical traffic traces of CDN server operation. Each trace could span a specific (large) number of requests or time and could be collected at a single server or could combine data across many servers. For the collected traces, we extract *features*, which include average requested object sizes, vector of inter-arrival times, and vector of stack distances [46]. We then form clusters of traces based on their features. Next, we associate an expert with a trace if its hit rate is within θ% of that of its best-performing expert. Cycling over each trace in each cluster gives us a map between features of a trace to a corresponding set of experts that are promising. This process aids the online operation of Darwin by potentially reducing the number of experts to be considered. Further, the θ threshold can help capture the best experts of similar traces that were not covered in the logs.

1b. Offline cross-expert predictors: We find that the performance of multiple experts that map to a cluster are correlated (as we explain later in Figure 5c), implying that we can predict the performance of an expert given that of another. To this end, we train neural networks for each ordered pair of experts – each one accepts trace features as input and outputs the conditional probability of hits of one expert given hits/misses of the other. These *cross-expert neural predictors* are used in the online phase below to predict the performance of all non-deployed experts using only the samples collected from the deployed expert (Section 4.1).

These offline steps are repeated periodically as more trace data is collected, resulting in more refined/new clusters and improved cross-expert prediction.

2. Online selection (Section 4.2): The online step proceeds over epochs of N_e requests each and consists of two phases: feature estimation, and best-expert identification and deployment.*(Feature Estimation)* The first $N_{warm-up}$ requests are used to estimate the features of the current traffic and thus associate the incoming traffic with a learned cluster (step 1a). *(Best-Expert Identification and Deployment)* From the corresponding set of experts with the learned cluster, we select a single best expert using a novel best-arm identification algorithm (Algorithm 1) with side information provided

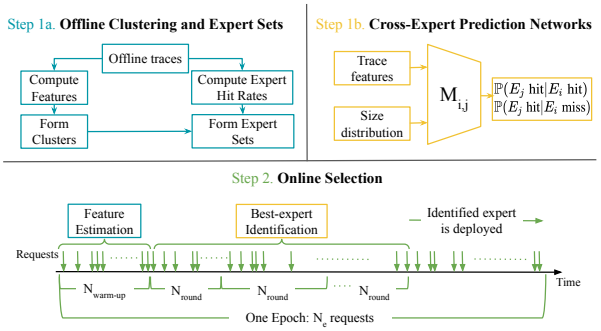


Figure 3: Darwin Workflow

by the cross-expert predictors (step 1b). The learned expert is then deployed for the remainder of the epoch.

4.1 Offline Training

The offline clustering and expert set association process is straightforward and we defer its description to Appendix A.1. We now discuss the cross-expert predictors.

The experts in Darwin share a structure: they promote all objects that occur with more than a threshold frequency and have a size smaller than a threshold in order to maximize HOC hit rates over a series of requests (traces). This structure can be leveraged in selecting experts.

To see why, let E_1 and E_2 be two experts characterized by tuples (f_1, s_1), (f_2, s_2) respectively. In any fixed trace, requests can be: *(a)* promoted by both E_1 and E_2, *(b)* promoted by only one of the two experts, or *(c)* never promoted by either expert. Specifically, objects that are requested at least $\max\{f_1, f_2\}$ times with size at most $\min\{s_1, s_2\}$ are of the type *(a)*, type *(c)* consists of requests with object size over $\max\{s_1, s_2\}$ or frequency lesser than $\min\{f_1, f_2\}$; the remainder are type *(b)*.

This suggests that the performance of these experts on a fixed trace is not independent of one another. It is thus reasonable to estimate the hit rate of one expert by observing the behavior of another on a fixed set of requests; we show empirical evidence in Figure 5c (Section 6).

Building on this idea, we train a 1-layer fully connected neural network $M_{i,j}$ for each ordered pair of experts E_i and E_j that belong to the same cluster-level best expert set. To train this network, we first extend the set of features associated with each trace with a bucketized version of its size distribution. This extended feature set is then used as a training point to train $M_{i,j}$, which predicts the conditional probabilities $\mathbb{P}(E_j \text{ hit}|E_i \text{ miss})$ and $\mathbb{P}(E_j \text{ hit}|E_i \text{ hit})$ over this trace. Adding the size distribution to the features helps provide sharper estimates of these conditional probabilities; the number of buckets to use can be chosen as necessary.

These conditional probability estimates also lead to estimates on variances: Let $\mathbb{P}(E_i\text{hit})$, $\mathbb{P}(E_i \text{ miss})$ denote the cache hit and miss frequencies of a fixed expert i on a given trace. For all experts $j \neq i$ the networks $M_{i,j}$ can be used to compute $V_{hit}(i,j) = \mathbb{P}(E_j \text{ hit}|E_i \text{ hit}) \cdot (1 - \mathbb{P}(E_j \text{ hit}|E_i \text{ hit}))$, the estimated variance of E_j hits given an E_i hit. Similarly, we can also compute $V_{miss}(i,j) = \mathbb{P}(E_j\text{hit}|E_i \text{ miss}) \cdot (1 - \mathbb{P}(E_j\text{hit}|E_i \text{ miss}))$ the estimated variance under E_i misses. Together, these can be combined to provide an estimate of the variance of hits of expert j given the performance of expert i using $\sigma^2_{ij} := \mathbb{P}(E_i \text{ hit}) V_{hit}(i,j) + \mathbb{P}(E_i \text{ miss}) V_{miss}(i,j)$. These variance computations will find use in the online expert identification stage below.

4.2 Online Expert Identification

We now move to the online deployment step of `Darwin`.

Feature Estimation. For the first $N_{warm-up}$ requests of an epoch of N_e requests, the user deploys an arbitrary expert (or one from the previous epoch) to instruct HOC admissions. It then computes the empirical features for this epoch based on the warm-up requests and then selects a small set of experts (Appendix A.1).

Best-Expert Identification and Deployment. Next, our (bandit) algorithm sequentially deploys different experts (from the set identified through feature learning) and collects rewards (HOC hit rates) from each expert. Using these collected rewards and fictitious reward samples generated using the cross-expert predictors (Section 4.1), the algorithm determines a single best expert at the end of "best-expert identification", and deploys this selected expert. There are two things to note: *First*, each expert deployment is over a series of N_{round} consecutive requests (termed as a *round*). At the end of a round, the next expert is deployed, and so on over rounds, until the end of the best-expert identification phase. The number of requests in a round, N_{round} is chosen to be sufficiently long such that the state of the cache (which has been determined by previously deployed experts) sufficiently de-correlates over time, and the reward estimates at the end of the round are representative of the currently deployed expert. *Second*, the number of *rounds* (equivalently, the number of experts) that are deployed in this phase is adaptively chosen online, to ensure that the expert chosen at the end of the phase is truly the best expert with a probability $\geq (1 - \delta)$, where, δ is an operator specified *failure probability*. This expert is then deployed for the remainder of the epoch.

From Experts to Bandits. The problem of expert selection is closely related to the Multi-armed Bandit problem with experts as arms and hit rates as rewards. In contrast to standard bandit feedback models where only rewards of experts *deployed* for HOC admissions can be observed, we can gather hit rate *estimates* for all experts using our cross-expert prediction models. There are, however, two things to note: *(a)*. The estimates of the non-deployed experts are only accurate if the networks predict the conditional hit rates reliably, *(b)*. Due to the randomness in the performance of the deployed expert, the reward estimates from the non-deployed experts are also random, with variances that depend on the deployed expert. This problem is one formulation of the Multi-armed bandit problem with Side Information; this form was introduced by [48], where the selected arm-dependent variances are encoded as a known side information matrix. Further, this information structure generalizes various structures studied in bandits including graph-based and full feedback; we refer to [4, 48] for details. However these works have focused on cumulative regret, whereas our objective is that of best-arm identification, which is quite different[3].

[3]Broadly, there are two bandit settings: (a) cumulative regret, where the objective is to continuously trade-off between exploration and exploitation of learned information throughout the deployment, and (b) best arm identification/pure exploration, where the goal is to determine the best expert at the end of a learning phase. The algorithms for these two settings can be quite different; we refer to [24] for additional discussion.

To the best of our knowledge, our work is the first to consider the best-arm identification setting with this richer feedback structure. This additional structure is utilized by our algorithm to identify the best expert in a finite expected time, whose scaling is *constant* with respect to the number of experts. This is in contrast to the classical setting, where the learning time typically grows with the number of experts.

Best Expert Identification with Side Information. We now make our setting formal: Let the cluster-level set identified after $N_{warm-up}$ samples have experts indexed by the set $[K] = \{1, 2, ..., K\}$. For any expert $i \in [K]$, let $\mu_i \in [0, 1]$ denote its mean hit rate (reward) and $\mu \in \mathbb{R}^K$ be the vector of mean rewards. We denote the variance of rewards observed from expert j when expert i is deployed (or played) by σ^2_{ij} and encode these variances in the matrix $\Sigma \in \mathbb{R}^{K\times K}$.

The learner first infers the side information matrix Σ by deploying each expert over a series of N_{round} requests using the performance prediction networks $M_{i,j}$ (see variance computation discussions in Section 4.1). Thereafter, in each round t, an expert E_t is chosen to be deployed in the HOC over N_{round} requests. At the end of the round, the learner computes the observed hit rate of expert E_t in this round and uses $M_{E_t,j}$ for all $j \neq E_t$ to form the vector of rewards $Y_t = (Y_1(t), Y_2(t), \ldots, Y_K(t))$. Here, $Y_{E_t}(t)$ is the true observed hit rate, while $Y_j(t), j \neq E_t$ are all fictitious samples generated by the prediction networks. We assume that for any expert i and any round t, $Y_i(t)$ is an independent Gaussian random variable with mean μ_i and variance $\sigma^2_{E_t i}$.

The learner maintains an estimate $\hat{\mu}_t$ of the mean rewards of each expert at the end of round t to instruct its future deployed experts. As in the standard best-arm identification problem of [19], we seek to design a triple (π, τ, ψ) of an expert sequence selection policy π, a stopping time τ and a recommendation rule ψ such that: $\mathbb{P}_{\mu,\Sigma,\pi}(\psi(\hat{\mu}_\tau) \neq i^*(\mu)) \leq \delta$. In words, the expert recommended using the rule ψ after τ rounds of running the policy π is the best expert $i^*(\mu) = \text{argmax}_{k\in[K]}\, \mu_k$ with a high probability of $1-\delta$. Let $\mathcal{E}(\Sigma)$ be the set of all mean reward vectors μ with Side information matrix Σ. Then, any triple (π, τ, ψ) satisfying the above condition for all $\mu \in \mathcal{E}(\Sigma)$ is said to be δ–sound [19, 24].

Track and Stop with Side Information. We propose the Track and Stop with Side Information algorithm (Algorithm 1) which builds on the well-known Track and Stop strategy [19] for best-arm identification with standard bandit feedback. It deploys experts to instruct HOC admissions sequentially over rounds and terminates once it has identified the best expert with a high probability $\geq 1-\delta$. This termination time is random and depends on the request sequence and statistics. Below, the algorithm estimates hit rates for each expert, determines the sequence in which experts are deployed our rounds, and stops when it is confident that it has identified the best expert.

Hit rate estimators: At each round t, the estimate $\hat{\mu}_i(t)$ for the mean reward of expert $i \in [K]$ is given by

$$\hat{\mu}_i(t) = \frac{1}{\rho_i(t)} \sum_{n=1}^{t} \frac{Y_i(n)}{\sigma^2_{E_n,i}}, \quad \rho_i(t) = \sum_{n=1}^{t} \frac{1}{\sigma^2_{E_n,i}} \tag{1}$$

Algorithm 1 Track and Stop with Side Information

1: Inputs: Side Information matrix Σ, failure probability $\delta > 0$, a function $\beta_t(\delta, \Sigma)$.
2: Choose each expert once and set $t = K, T_i(K) = 1$.
3: **while** $Z_t < \beta_t(\delta, \Sigma)$, **do**
4: Compute $\alpha^*(\hat{\mu}_t, \Sigma)$ defined in Equation (3).
5: Deploy $E_{t+1} = \text{argmax}_{i\in[K]}\, t\alpha^*_i(\hat{\mu}(t), \Sigma) - T_i(t)$.
6: Observe the reward vector Y_t.
7: Update estimates $\hat{\mu}_i(t+1)$ using Equation (1).
8: Increment the counter of expert plays $T_{E_{t+1}}$.
9: Compute $Z_{t+1} = \Phi(\hat{\mu}_t, T(t+1))$, Φ as in Equation (2).
10: **end while**
11: **return** $\psi(\hat{\mu}(t)) = \text{argmax}_{i\in[K]}\, \hat{\mu}_i(t)$

The above is simply a modification to the standard empirical mean and is obtained by re-weighting each sample according to its corresponding variance and then normalizing the estimate using $\rho_i(t)$. It was previously used by [4] to design asymptotically optimal algorithms for the cumulative regret setting of [48] when the rewards are Gaussian.

Expert sequence selection policy π: To determine the sequence of experts to deploy, the learner repeatedly (at the beginning of each round) solves the optimization problem:

$$\Phi(\nu, \alpha) = \inf_{\nu' \in \mathcal{E}_{alt}(\nu)} \sum_{i=1}^{K} \alpha_i \sum_{j=1}^{K} \frac{(\nu_j - \nu'_j)^2}{2\sigma^2_{i,j}} \tag{2}$$

$$\alpha^*(\nu, \Sigma) = \underset{\alpha\in\mathcal{P}_{K-1}}{\text{argmax}}\, \Phi(\nu, \alpha). \tag{3}$$

The cost function in Φ is derived from a KL divergence between two Gaussian distributions. Intuitively, if ν_j is replaced with the empirically learned mean, this cost quantifies the most likely Gaussian distribution from among the alternate environments that could have resulted in the observed samples. Thus, in some sense, this measures the distance to the 'most likely' wrong expert that we could end up with. Here, $\mathcal{P}_{K-1}$ is the probability simplex in K dimensions and $\mathcal{E}_{alt}(\nu) = \{\nu' \in \mathcal{E}(\Sigma) : i^*(\nu) \cap i^*(\nu') = \emptyset\}$ is the set of *alternate environments*. Recall that $\mathcal{E}(\Sigma)$ is the collection of all possible environments; for each μ in this set, $\mathcal{E}_{alt}(\mu)$ is the subset of environments that do not share the same index for the best expert. For a given mean vector ν and a side information matrix Σ, the solution $\alpha^*(\nu, \Sigma)$ is a probability distribution over experts. In each round t, the learner solves for $\alpha^*(\hat{\mu}_t, \Sigma)$. Intuitively, α^* corresponds to the optimal fraction of rounds that each expert should have been deployed until now, assuming that the current empirical estimates are correct. Thus, in the current round, the algorithm deploys that expert which is the most under-deployed expert with respect to this estimated distribution (Line 5, Algorithm 1).

Stopping time τ: The algorithm computes the 'information level' $Z_t = \Phi(\hat{\mu}_t, T(t))$. Here, $T_i(t)$ is the number of times that expert i has been deployed up to time t, and $T(t)$ is the vector of these deployment counts. At the beginning of each round, if this information level exceeds the given threshold function $\beta_t(\delta, \Sigma)$ (specified in Theorem 1), the algorithm terminates (thus defining the stopping time).

Expert recommendation rule ψ: The returned expert is simply the most empirically promising one at the time of stopping. Formally, $\psi(\hat{\mu}_\tau) = \text{argmax}_{i\in[K]}\, \hat{\mu}_i(\tau)$

Theoretical Results. Our setting differs from the standard bandit feedback due to the fictitious samples we gather from the cross-expert predictors. These are explicitly used in our estimators in Equation (1) and in order to guarantee that Algorithm 1 is $\delta-$sound, we require concentration bounds on the performance of these estimators. The difficulty is that the series of expert deployments is random due to the expert sequence selection policy π. To overcome this, we develop *anytime concentration bounds* for these scaled empirical estimators that do not depend on the selection strategy used to pick experts. For this, we analyze a martingale process, both over time and experts. Beyond proving soundness, the martingale evolution over the pair (experts, time) – as opposed to only time – is crucial to show that the stopping time of our algorithm is bounded by a quantity that *does not scale* with the number of experts K. As observed before, this is an expected, yet interesting artifact of the increased feedback due to correlation across experts.

We now state our theoretical guarantees. The first result shows that the proposed policy, stopping criterion and expert recommendation rule form a $\delta-$sound triple.

Theorem 1. *Let $\sigma^2_{min}, \sigma^2_{max}$ be the minimum and maximum variances in Σ respectively. Let M be such that the rewards $Y_i(t) \in [-M, M]$ with probability at least $1-\delta/2$ for all $i \in [K], t \in \mathbb{N}$. Using $\kappa = \frac{\sigma^2_{min}}{\sigma^2_{max}}$ and $\beta_t(\delta, \Sigma) = \frac{Kt}{2\kappa} + \frac{KM^2}{2\sigma^2_{min}\kappa\sqrt{C}}\sqrt{t\log(2/\delta)}$ for a constant C, the triple (π, τ, ψ) suggested by Algorithm 1 is $\delta-$sound.*

The next result characterizes the expected stopping time.

Theorem 2. *Let ν be any environment with a unique best expert. Then, with expert sequence selection policy π and stopping time τ as in Algorithm 1, it holds that*

$$\lim_{\delta\to 0}\frac{\mathbb{E}_{\nu\pi}[\tau]}{\log(1/\delta)} = \left(\frac{M}{\sigma^2_{min}}\cdot\frac{K}{2\kappa\Phi(\nu,\alpha^*(\nu,\Sigma))-K}\right)^2.$$

We recall that $\Phi(\mu,\alpha), \alpha^*(\nu,\Sigma)$ are defined in Equations (2) *and* (3) *respectively.*

Complete proofs of both these theorems can be found in Appendix A.2. The key observation here is that the limit in Theorem 2 does not scale in the number of experts K, whereas with standard bandit feedback, the corresponding limit scales linearly in the number of experts.

Remark: We choose to develop the best expert selection algorithm for Gaussian rewards to ease presentation. Specifically, this assumption allows us to write the quadratic-form inner summations (which are KL divergences between two Gaussians) in Equation 2 and also provide a unique closed-form solution to the optimization problem in Equation 3. This solution is then used to establish Theorem 2. This provides us with intuition on the scaling of the stopping time with the total number of experts K. Note that we do not require the Gaussian rewards to prove soundness in Theorem 1. With non-Gaussian rewards, our approach can be used by appropriately modifying Equation 2 and numerically solving Equation 3.

5 IMPLEMENTATION

We implemented Darwin on top of ATS [18]. The original ATS consists of a RAM cache and a disk cache. We inject the Darwin admission policy by modifying the conditions for an object to be added to the RAM cache to include a frequency and a size threshold configured by the user. We make further modifications to support the online phase of Darwin.

To aid in this, we track the request counts on the cache server. Once the request count reaches an action point (e.g., the end of the feature collection stage, or the end of each bandit round), our prototype starts a new thread to perform Darwin's additional operations. At the end of the feature collection stage (which is lightweight due to the optimizations discussed in Section 6.4), the thread looks up the cluster and loads the corresponding best experts into memory. At the end of each bandit round, we calculate the rewards of this round and select the next round's arm (expert) in parallel with the cache processing. Once the new expert decisions are made, we change the threshold values of the HOC admission policy.

We also implemented a Darwin simulator based on the LRB simulator [39] and the feature extraction module [34], which simulates a two-level cache hierarchy.

6 EVALUATION

We evaluate Darwin using both simulations and prototype experiments. We seek to answer four main questions:
(i) How well can Darwin adapt to traffic changes and improve CDN caching OHR performance compared to fixed experts and SOTA learning-based approaches? (Section 6.1)
(ii) How well do Darwin components of clustering, cross-expert predictors, and online selection perform? (Section 6.2)
(iii) How well does Darwin work toward optimizing different metrics? (Section 6.3)
(iv) What is the overhead of using Darwin, in terms of latency, throughput, CPU, and memory usage? (Section 6.4)

We now discuss our methodology and setup.
Simulator Setup. We build and run the simulator on a single server. The server's HOC size is configured to be 100MB, and the disk size is 10GB. We also experiment with larger cache sizes, specifically, 200MB and 500MB. We are limited by our access to computational resources to explore even larger cache sizes. However, our study with 100MB, 200MB, and 500MB caches shows that Darwin's performance benefits hold for servers equipped with larger caches.
Testbed Setup. We set up the client, proxy (CDN server), and origin servers in Cloudlab [16]. Each node has a 16-core AMD 7302P @ 3.00GHz, 128GB ECC Memory (8x 16 GB 3200MT/s RDIMMs), two 480 GB 6G SATA SSD, and two dual-port Mellanox ConnectX-5 25Gb GB NIC (PCIe v4.0). Each node pair of client-proxy and proxy-origin is connected with a 20Gbps bandwidth link. We inject a latency of 10ms between the client and proxy and 100ms between the proxy and origin. By default, we set the cache RAM size to 100MB, and disk size to 1024GB.
CDN Traces. We first describe how we generate traces to study Darwin's benefits for a 100MB cache size. We then describe how we "scale" these traces to study the benefits for CDN servers using larger cache sizes of 200MB and 500MB.

We employ two sets of CDN traces in the experiments: an offline training set used to train the cross-expert prediction models and a testing set for online testing. To create a large and diverse dataset, we generate synthetic traces based on the Download and Image traces with various mixed ratios using Tragen [34]. The sum of the

request rates for the two traffic classes in the production trace is 265.9 req/s. We vary the proportion of these two traffic classes, from 100:0 to 0:100, creating 100 mixed request rate configurations in total. We generate 10 traces with 10M requests for each configuration. 7 of these are added to the training set (offline train trace), and the rest 3 traces are used for model testing (offline test trace). We also create one trace for each configuration that contains 100M requests to be used for the end-to-end evaluations (online test trace). For all the traces, the first 1M of requests are used as "cache warmup" requests in the trace, and the statistics of these requests are not counted in the final results.

For 200MB and 500MB cache sizes, instead of using the same traces as above, we scale up the object sizes of the 100MB traces by 2× and 5×, respectively, and additionally perturb each object's size randomly by ±20% to synthetically generate "new" traces. Our rationale for scaling the traces is that in a production setting, the load balancer of the CDN would assign more traffic to servers with larger caches. For example, servers equipped with larger caches will typically serve larger volumes of traffic with higher request rates and/or larger objects.

Baselines. In the simulation experiments, we compare with StaticExpert(s), AdaptSize (AS) (Section 3.2.1), DirectMapping (Direct) (Section 4), Percentile (P), and HillClimbing (HC-Δ_s). Each StaticExpert is a combination of a frequency threshold (f=2-7) and a size threshold (s=10, 20, 50, 100, 500, 1000kB). We scale up the size thresholds for the larger cache sizes.

Percentile (P) works as follows: In N-request windows, we update the empirical distributions of frequencies and sizes of incoming requests. For the next N requests, it deploys the expert (f, s) with f, s closest to the $60^{th}, 90^{th}$ percentiles (respectively) of the empirical distribution hitherto. We use $N = 100K$ requests and the percentile values are picked to be the best-performing ones for this window size.

For HillClimbing, the learner deploys an expert (f, s) in the main cache for N requests and concurrently runs two shadow caches; one each for experts $(f + \Delta_f, s), (f, s + \Delta_s)$. It then updates the main cache with the best-performing expert of the three. When the expert deployed in the main cache does not change, the shadow caches are updated to run $(f - \Delta_f, s), (f, s - \Delta_s)$. In the above we use, $\Delta_f = 1$ and $\Delta_s = \{1KB, 10KB\}$ and $N = 0.5M$ in our evaluation.

In the prototype experiments, we compare with the same set of static experts on ATS. Unless otherwise stated, we configure Darwin with $\theta = 1\%$, $N_e = 100M$, $N_{warmup} = 3M$ and $N_{round} = 0.5M$.

Metrics. For simulation, we consider the *OHR*, the linear combination $OHR + k * disk\_writes$, and byte miss ratio (BMR) as the objectives for HOC admissions. For the prototype, we also measure the request first-byte latency, server throughput, CPU and memory use, and network throughput.

6.1 Robustness to Traffic Changes

We evaluate the OHR of Darwin and multiple baselines in the simulator using our online test trace set. We find that Darwin outperforms baselines by 3%-43%. While the lower-range improvement numbers seem unimpressive at face value, we remind the reader that even minor improvements in hit rates translate to significant reductions in network bandwidth usage that leads to improvements. Since a large CDN could incur a midgress of tens of Tbps at a cost of tens of millions of dollars per year, even a small midgress bandwidth reduction due to improved hit rates translates into large cost savings for the CDN [42].

Comparison with static baselines. To illustrate the performance of Darwin and the baseline algorithms against changing traffic, we pick an ensemble set made up of traces with a variety of best static experts. We group the online test traces by their best static experts and randomly pick one trace from each group to add to the ensemble set. Figure 4a shows the distribution of Darwin's HOC OHR improvement rates against each baseline (more complete baseline results in Appendix A.3). Note that each trace is with a stable traffic mixture, and therefore a suitable static expert can outperform Darwin in one trace as Darwin runs with a suboptimal expert in a proportion of the requests when learning. But no static expert works well in all traces.

Comparison with adaptive methods. Darwin adapts to traffic changes better than the previous state-of-the-art adaptive methods described below.

Percentile: Fixed percentile thresholds are non-optimal for a proportion of the traces and therefore have worse hit rates for those traces. In our experiments, Percentile is 10% worse on average than Darwin.

HillClimbing: In our experiments, HillClimbing has access to a larger slew of experts (e.g., all experts with frequency thresholds that are multiples of Δ_f) than Darwin (limited to the given expert set). Further, HillClimbing is provided with additional computational resources in the form of two shadow caches which are used to determine the expert swaps (one each for size and frequency). However, even with these significant advantages, it under-performs Darwin by $\geq$ 3% on average. Further, HillClimbing also requires careful tuning of the jump sizes (Δ_f, Δ_s), as well as the number of requests after which experts are switched. And, it also suffers from the pitfall of displaying suboptimal performance due to the presence of local optima.

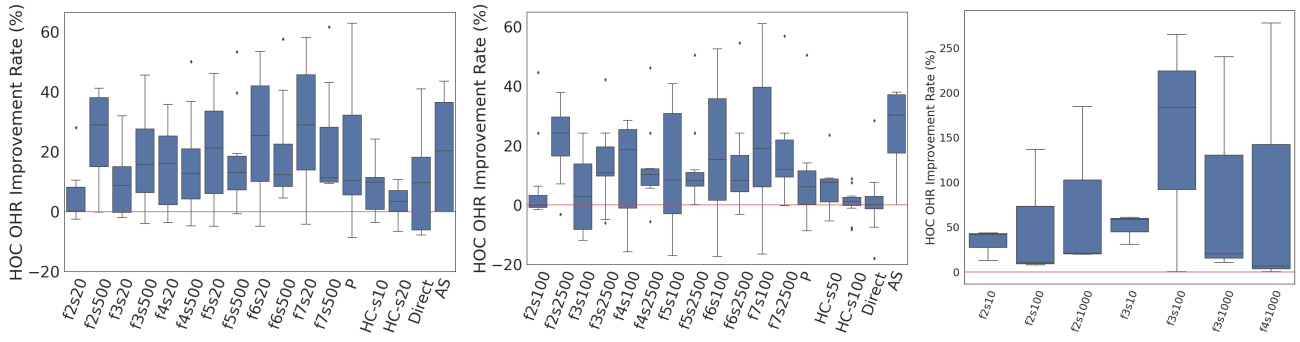
AdaptSize: As noted previously, AdaptSize examines only one dimension for HOC admissions, the size, which is insufficient as the cache can be polluted by, e.g., many infrequently requested objects (it is 20% worse than Darwin on average).

DirectMapping: Direct mapping from traffic features to *the single best (f, s) threshold configurations* also performs 7% worse than Darwin on average. This is because it is not robust to errors in feature collection or in the process of learning the mapping; Darwin, due to directly testing and then selecting among *multiple good candidates* can better accommodate any potential errors in feature collection, clustering, etc.

We observe similar results with a larger cache size (Figure 4b) as well as in our prototype with a subset of static experts (Figure 4c).

6.2 Effectiveness of Darwin Components

How quickly can the features reach reliable values? The feature estimation phase at the beginning of each epoch in Darwin's online step is crucial as it decides the expert set to be considered; errors here impact performance significantly. To choose the number of requests used in this phase ($N_{warm-up}$) appropriately, we study the convergence of empirical features to their true values. We



(a) Simulation (100MB) **(b) Simulation (500MB)** **(c) Prototype**

Figure 4: OHR for Darwin vs. Baselines (P = Percentile, HC = HillClimbing, Direct = DirectMapping, AS = Adaptsize). (a) Simulation results with 100MB HOC size. (b) Simulation results with 500MB HOC size. We observed similar results with a 200MB cache size. We omit the results for brevity. (c) Prototype results with low concurrency.

use average size (size_avg), the first 7 average inter-arrival times (iat_avg's), and stack distances (sd_avg's) as features. Figure 5a uses the 10M length offline traces and computes the true features using all 10M requests. We compare these true features with the first xM requests of the trace (x varies as in the legend) and plot the %-age difference in absolute value. We see that feature values converge to within a 10% error margin using only the first 3M requests. Importantly, this trend – of needing 3M requests – also holds for the 100M length online test traces. We use this value for $N_{warm-up}$ from here on. $N_{warm-up}$ for the online test trace thus is just 3% of the total number of requests (complete result in Appendix A.3).

Does clustering and expert set formation help? We present evidence of our offline clustering leading to a reduction in the number of considered experts. We begin with 36 experts and use the offline traces to perform clustering (into 52 clusters) and expert set association. Recall that the threshold θ induces diversity by considering all experts within θ% of the best-performing expert of an offline training trace to be part of its promising experts set. We observe that for every offline test trace and our online test traces, at least one of the trace's best experts is always included in its corresponding expert set for varying thresholds θ (detailed result in Appendix A.3). With $\theta = 1$ (our default choice), we observe an 82% reduction in the number of experts on average; even with $\theta = 5$, we can extract a 35% reduction. The CDF of the number of experts that remain after clustering is in Figure 5b.

We also created experts with three decision knobs: frequency, size, and recency, and formed clusters and expert sets using the offline training traces. Here too, we saw a 90% reduction in the number of experts using a $\theta = 1\%$ threshold, resulting in only a few experts that Darwin's online algorithm needs to work with (results deferred to Appendix A.3).

How accurate are the cross-expert predictors? For our online identification process, it is sufficient for our prediction networks to accurately estimate the *ordering* of expert hit rates. We say two experts are 'k% proximal' on a trace if their hit rates on this trace are less than k% apart. For each predictor $M_{i,j}$, we compute the fraction of traces on which the experts i, j are either proximal or $M_{i,j}$ predicts the same ordering between hit rates of experts i and j as the ground truth. We refer to this fraction as the *order prediction accuracy* and its CDF over all 1260 cross-expert predictors (formed using the 36 experts) versus proximity k is shown in Figure 5c. Even with the strictest 1% proximality, more than 90% of the predictors reach > 80% order prediction accuracy (across our test data points). In Appendix A.3, we show that our cross-expert predictors also work well with test traces drawn from a different distribution than the one they were trained on.

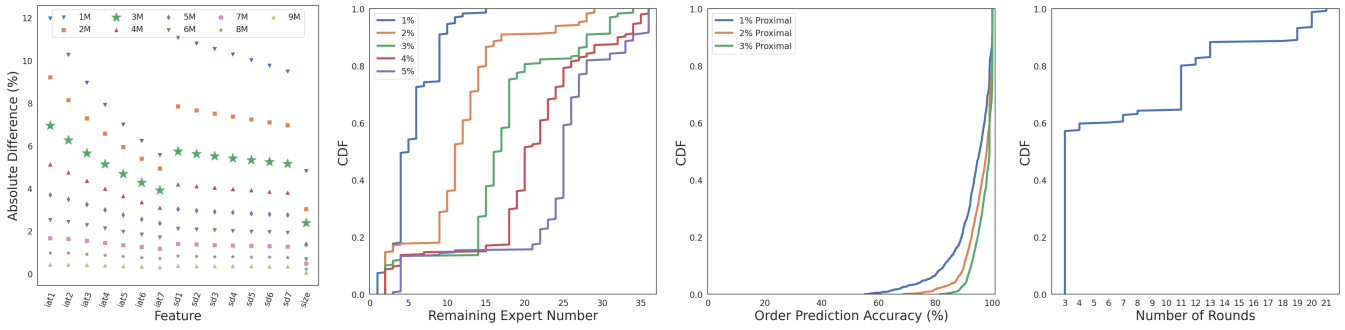
How many rounds for online expert identification? We observe that the average hit rates for the candidate experts converge fast for most of the traces in the online identification stage, and we can quickly identify a consistent best expert from them. For a trace running the online expert identification, we say that it has found the best expert if either the bandit algorithm terminates or an expert is consistently selected by the bandit for 5 consecutive rounds. We track the number of rounds from the start of the bandit stage until the best expert is found. Figure 5d illustrates that starting from the 12th round onwards, $\geq$ 80% of our online test traces achieve stability with the best expert (requiring $\leq$5.5M requests for convergence). The worst-case scenario for convergence spans 21 rounds (equivalent to 10M requests, constituting 10% of the trace).

6.3 Support for Other Goals

We show that Darwin can be easily customized to other objectives. For a new optimization metric, we need two slight modifications: 1. retrain the cluster best expert mapping based on the new metric results of the experts; 2. use the new metric as the reward in the online phase.

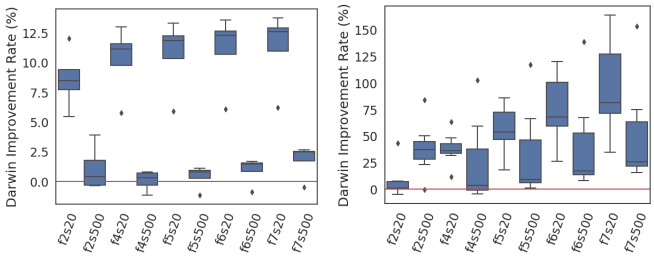
We first target minimizing the BMR of the HOC to reduce the bytes written to the DC or to the origin server. To estimate the unobserved experts' BMR performance, we perform a simple calculation based on the observed bucketized size distribution and the output of the existing OHR cross-expert predictors. Figure 6a shows that Darwin reduces the HOC Byte Miss Ratio by 0.37%-11.28%.

Another objective that we experiment with is a combination of HOC OHR and disk writes. We seek to maximize $OHR - \frac{DiskWrite}{\#Requests}$. We approximate the disk write bytes to be the bytes missed in HOC. With that, we can calculate HOC OHR and disk writes respectively with the existing OHR cross-experts and the bucketized size distribution. Figure 6b shows that Darwin improves the metric by 7.47%-96.67%.



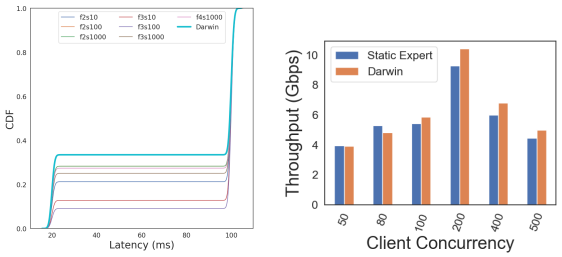
(a) Feature Convergence (b) Improvement from Clustering (c) Cross-expert Prediction Accuracy (d) Bandit Rounds for Identification

Figure 5: Effectiveness of Darwin Components: (a) Feature convergence using first xM requests; (b) CDF of the number of remaining experts after clustering for different θ; (c) Order prediction accuracy for our cross-expert predictors; (d) Number of rounds required for best expert-identification.



(a) HOC BMR (b) HOC OHR-Disk Write/#Request

Figure 6: Darwin for other objectives.



(a) First Byte Latency (b) Throughput

Figure 7: Darwin prototype's latency and throughput performance

As we have alluded to before, existing baselines such as Hill Climbing and AdaptSize cannot readily adapt to objectives such as disk writes that are hardware-dependent and are therefore complex to model and simulate. Hill Climbing uses the notion of a shadow cache to simulate and derive the hit rate of an incrementally larger cache. Likewise, AdaptSize uses a Markov chain model to simulate object accesses and derive OHR as a function of the size threshold. The models simulated in both the above approaches are explicitly tailored to deriving hit rates. It is not clear how these models can be extended to other more complex hardware-dependent objectives. In contrast, Darwin does not rely on a specific modeling and simulation approach for cache performance and can work for objective functions that are hard to explicitly model and simulate.

6.4 Overhead

In our prototype, we conduct measurements of performance and resource usage overheads. We demonstrate the positive impact of Darwin on throughput and latency through the increased rate of cache hits. Additionally, we find that Darwin's implementation minimally impacts CPU and memory utilization.

Response Latency. Figure 7a shows the latency CDF for a concatenated trace that consists of four 100M online test traces with different best experts. We observe that Darwin doesn't impose additional latency overhead, and improves the first-byte latency by reducing the requests forwarded to the origin server (on account of its better OHR). All the Darwin components – e.g., feature collection, cluster lookup, and looking up prediction networks – create a new thread to perform the work in parallel. Thus, the learning logic is not in the critical path of cache processing.

Throughput. Figure 7b shows the peak of the application throughput of Darwin across multiple concurrency levels. Higher concurrency can increase throughput but also increases synchronization costs (lock contention for the HOC). We compare Darwin with the static ($f = 2, s = 2K$) expert. In both cases, the sweet spot for throughput vs synchronization overhead is around 200 requests. Darwin is able to reach an average throughput of 10.4Gbps for 200 client threads (static expert reaches 9.3Gbps).

For low concurrency levels, Darwin's throughput is comparable to the static expert, but Darwin's OHR (which stays unaffected at these concurrency levels, but isn't shown) is significantly better as shown in Figure 4c. At higher concurrency, Darwin's hit rates are inferior to that at low concurrency (this trade-off has been observed in prior works [10]), as are the static experts'. But Darwin offers better throughput – this is because, on account of its better OHR, Darwin is able to skip round trips to the origin server.

CPU and Memory. Without Darwin, the average CPU usage is 2% to process the requests in real-time. The peak CPU usage with Darwin can reach 99.2%. But this only happens instantaneously in two specific steps: 1. when Darwin classifies the features and does cluster lookup at the end of the feature collection stage; 2. when Darwin infers from the cross-expert predictors at the beginning of each round. The instances of high CPU use are infrequent relative to overall request processing at the cache and are amortized out; as such, we see no perceptible increase in average CPU use.

Darwin's memory overhead is also reasonable, with a peak of 4GB. During the feature collection stage, we create a tree structure to extract the stack distances and inter-arrival times of the objects. This tree is deleted at the end of the stage, and we only store a

single feature vector with 15 entries. During the online selection stage, the largest memory usage is for the cross-expert prediction networks. To avoid the performance overhead of loading the prediction networks repeatedly, we keep them in memory for the whole online selection stage, which contributes to the 4GB memory use. The other data structures stored include the following, and they also pose minimal overhead: (1) bucketized size distribution, whose entry number is the same as the size threshold selection range; (2) model variance matrix, whose entry number is the number of expert pairs; (3) reward vector, whose entry number is the number of experts. For 2 and 3, we only consider the experts from the cluster outputs, so it's significantly smaller than the original expert space.

7 RELATED WORK

Section 2 discussed related work on heuristic-based policies for HOC admission and eviction, as well as learned HOC admission policies. Here, we cover other related works, spanning bandit algorithms and learned cache eviction.

Bandit Algorithms. Prior work on bandits studied several types of side information by imposing additional structure on the space of actions. These include graph-structured actions [1, 11, 43], latent causal confounding [25, 38, 51], and noisy side observations [4, 48] among others. These are mostly in the setting of cumulative regret. Best-arm identification with side information has been considered for the causal confounding case in [23, 37], and more recently for linear bandits in [26]. Our setting in Section 4.2 is that of best-arm identification with noisy side observations as in [4, 48].

Learned Cache Eviction. Recent works have proposed learning-based CDN cache eviction policies. LFO [9], LRB [39], CACHEUS [33], LeCaR [45], LHD [6], and DeepCache [30] all use learning or prediction models to decide object eviction. Almost all of these approaches are designed to optimize a single performance objective like OHR. While Darwin focuses on studying HOC admissions, we argue that our approach can be flexibly extended to learn CDN eviction decisions with multiple objectives; we leave a systematic exploration for future work.

8 CONCLUSION

We presented Darwin, a CDN cache management system that uses a novel cache admission approach. Darwin is robust to traffic pattern changes, can optimize different caching objectives, and accommodates unrestricted action spaces. Darwin's offline clustering and expert prediction approaches provide crucial side information to its online phase, where a bandit selection algorithm quickly selects the right admission policy to use for the currently observed traffic pattern. Our evaluation shows that Darwin is highly effective at adaptation and at optimizing both hit rates and operational costs (such as disk writes and BMR) while offering high throughput at low overhead. Besides cache management, our novel learning paradigm of offline clustering and online expert selection is likely applicable to auto-tuning other system components and is the subject of future research.

Ethics Statement: Our work uses synthetic traces or anonymized production CDN traces and raises no ethical concerns.

ACKNOWLEDGEMENT

We would like to thank our shepherd, Katerina Argyraki, and the anonymous SIGCOMM reviewers for their invaluable feedback. This work was supported in part by the National Science Foundation under Grants CCF-2019844, CNS-2207317, CNS-2112471, CNS-2106299, CNS-1763617, and the Machine Learning Lab (MLL) at UT Austin.

REFERENCES

[1] Amin, K., Kearns, M., and Syed, U. Graphical models for bandit problems. *arXiv preprint arXiv:1202.3782* (2012).
[2] Ari, I., Amer, A., Gramacy, R. B., Miller, E. L., Brandt, S. A., and Long, D. D. Acme: Adaptive caching using multiple experts. In *WDAS* (2002), vol. 2, pp. 143–158.
[3] Asiso. Overview of the cdn akamai. https://www.asioso.com/en/blog/overview-of-the-cdn-akamai-b520, Feb 2021.
[4] Atsidakou, A., Papadigenopoulos, O., Caramanis, C., Sanghavi, S., and Shakkottai, S. Asymptotically-optimal gaussian bandits with side observations. In *International Conference on Machine Learning* (2022), PMLR, pp. 1057–1077.
[5] Basat, R. B., Einziger, G., Friedman, R., and Kassner, Y. Randomized admission policy for efficient top-k and frequency estimation. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications* (2017), IEEE, pp. 1–9.
[6] Beckmann, N., Chen, H., and Cidon, A. LHD: improving cache hit rate by maximizing hit density. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018* (2018), S. Banerjee and S. Seshan, Eds., USENIX Association, pp. 389–403.
[7] Belady, L. A. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal 5*, 2 (1966), 78–101.
[8] Belady, L. A. A study of replacement algorithms for a virtual-storage computer. In *IBM Systems journal* (1996), vol. 5, pp. 78–101.
[9] Berger, D. S. Towards lightweight and robust machine learning for CDN caching. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks, HotNets 2018, Redmond, WA, USA, November 15-16, 2018* (2018), ACM, pp. 134–140.
[10] Berger, D. S., Sitaraman, R. K., and Harchol-Balter, M. 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.
[11] Buccapatnam, S., Eryilmaz, A., and Shroff, N. B. Stochastic bandits with side observations on networks. In *The 2014 ACM international conference on Measurement and modeling of computer systems* (2014), pp. 289–300.
[12] Chen, F., Sitaraman, R. K., and Torres, M. End-user mapping: Next generation request routing for content delivery. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 167–181.
[13] Cherkasova, L., and Ciardo, G. Role of aging, frequency, and size in web cache replacement policies. In *High-Performance Computing and Networking: 9th International Conference, HPCN Europe 2001 Amsterdam, The Netherlands, June 25–27, 2001 Proceedings 9* (2001), Springer, pp. 114–123.
[14] Dilley, J., Maggs, B. M., Parikh, J., Prokop, H., Sitaraman, R. K., and Weihl, W. E. Globally distributed content delivery. *IEEE Internet Computing 6*, 5 (2002), 50–58.
[15] Dubhashi, D. P., and Panconesi, A. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
[16] Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., et al. The design and operation of cloudlab. In *USENIX Annual Technical Conference* (2019), pp. 1–14.
[17] Einziger, G., Friedman, R., and Manes, B. Tinylfu: A highly efficient cache admission policy. *ACM Transactions on Storage (ToS) 13*, 4 (2017), 1–31.
[18] Foundation, T. A. S. Apache traffic server. https://trafficserver.apache.org/, 2018. Accessed: 2023-01-30.
[19] Garivier, A., and Kaufmann, E. Optimal best arm identification with fixed confidence. In *Conference on Learning Theory* (2016), PMLR, pp. 998–1027.
[20] Guan, Y., Zhang, X., and Guo, Z. Caca: Learning-based content-aware cache admission for video content in edge caching. In *Proceedings of the 27th ACM International Conference on Multimedia* (2019), pp. 456–464.
[21] Jeong, J., and Dubois, M. Cost-sensitive cache replacement algorithms. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* (2003), IEEE, pp. 327–337.
[22] Kirilin, V., Sundarrajan, A., Gorinsky, S., and Sitaraman, R. K. Rl-cache: Learning-based cache admission for content delivery. In *Proceedings of the 2019 Workshop on Network Meets AI & ML* (2019), pp. 57–63.
[23] Lattimore, F., Lattimore, T., and Reid, M. D. Causal bandits: Learning good interventions via causal inference. In *Advances in Neural Information Processing Systems* (2016), pp. 1181–1189.
[24] Lattimore, T., and Szepesvári, C. *Bandit algorithms*. Cambridge University

Press, 2020.
[25] Li, L., Chu, W., Langford, J., and Schapire, R. E. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web* (2010), pp. 661–670.
[26] Li, Z., Ratliff, L., Nassif, H., Jamieson, K., and Jain, L. Instance-optimal pac algorithms for contextual bandits. *arXiv preprint arXiv:2207.02357* (2022).
[27] Maggs, B. M., and Sitaraman, R. K. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review 45*, 3 (2015), 52–66.
[28] McAllister, S., Berg, B., Tutuncu-Macias, J., Yang, J., Gunasekar, S., Lu, J., Berger, D. S., Beckmann, N., and Ganger, G. R. Kangaroo: Theory and practice of caching billions of tiny objects on flash. *ACM Trans. Storage 18*, 3 (2022), 21:1–21:33.
[29] McDiarmid, C. Concentration. *Probabilistic methods for algorithmic discrete mathematics* (1998), 195–248.
[30] Narayanan, A., Verma, S., Ramadan, E., Babaie, P., and Zhang, Z. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML, NetAI@SIGCOMM 2018, Budapest, Hungary, August 24, 2018* (2018), ACM, pp. 48–53.
[31] Nygren, E., Sitaraman, R. K., and Sun, J. The Akamai Network: A platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.
[32] Rizzo, L., and Vicisano, L. Replacement policies for a proxy cache. *IEEE/ACM Transactions on networking 8*, 2 (2000), 158–170.
[33] Rodriguez, L. V., Yusuf, F. B., Lyons, S., Paz, E., Rangaswami, R., Liu, J., Zhao, M., and Narasimhan, G. Learning cache replacement with cacheus. In *FAST* (2021), pp. 341–354.
[34] Sabnis, A., and Sitaraman, R. K. Tragen: a synthetic trace generator for realistic cache simulations. In *Proceedings of the 21st ACM Internet Measurement Conference* (2021), pp. 366–379.
[35] Sazoglu, F. B., Cambazoglu, B. B., Ozcan, R., Altingovde, I. S., and Ulusoy, Ö. A financial cost metric for result caching. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval* (2013), pp. 873–876.
[36] Schomp, K., Bhardwaj, O., Kurdoglu, E., Muhaimen, M., and Sitaraman, R. K. Akamai dns: Providing authoritative answers to the world's queries. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 465–478.
[37] Sen, R., Shanmugam, K., Kocaoglu, M., Dimakis, A., and Shakkottai, S. Contextual bandits with latent confounders: An nmf approach. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (2017), pp. 518–527.
[38] Sharma, N., Basu, S., Shanmugam, K., and Shakkottai, S. On under-exploration in bandits with mean bounds from confounded data. *arXiv preprint arXiv:2002.08405* (2020).
[39] Song, Z., Berger, D. S., Li, K., and Lloyd, W. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation* (2020).
[40] Suksomboon, K., Tarnoi, S., Ji, Y., Koibuchi, M., Fukuda, K., Abe, S., Motonori, N., Aoki, M., Urushidani, S., and Yamada, S. Popcache: Cache more or less based on content popularity for information-centric networking. In *38th Annual IEEE conference on local computer networks* (2013), IEEE, pp. 236–243.
[41] Sundarrajan, A., Feng, M., Kasbekar, M., and Sitaraman, R. K. 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* (2017), pp. 55–67.
[42] Sundarrajan, A., Kasbekar, M., Sitaraman, R. K., and Shukla, S. Midgress-aware traffic provisioning for content delivery. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)* (July 2020), USENIX Association, pp. 543–557.
[43] Valko, M., Munos, R., Kveton, B., and Kocák, T. Spectral bandits for smooth graph functions. In *International Conference on Machine Learning* (2014), pp. 46–54.
[44] Vietri, G., Rodriguez, L. V., Martinez, W. A., Lyons, S., Liu, J., Rangaswami, R., Zhao, M., and Narasimhan, G. Driving cache replacement with ml-based lecar. In *HotStorage* (2018), pp. 928–936.
[45] Vietri, G., Rodriguez, L. V., Martinez, W. A., Lyons, S., Liu, J., Rangaswami, R., Zhao, M., and Narasimhan, G. Driving cache replacement with ml-based lecar. In *10th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2018, Boston, MA, USA, July 9-10, 2018* (2018), A. Goel and N. Talagala, Eds., USENIX Association.
[46] Wikipedia. Stack distance, 2023. https://en.wikipedia.org/wiki/Cache_performance_measurement_and_metric.
[47] Wu, K.-L., Yu, P. S., and Wolf, J. L. Segment-based proxy caching of multimedia streams. In *Proceedings of the 10th international conference on World Wide Web* (2001), pp. 36–44.
[48] Wu, Y., György, A., and Szepesvári, C. Online learning with gaussian payoffs and side observations. *Advances in Neural Information Processing Systems 28* (2015).
[49] Yan, G., Li, J., and Towsley, D. Learning from optimal caching for content delivery. In *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Virtual Event, Munich, Germany, December 7 - 10, 2021* (2021), G. Carle and J. Ott, Eds., ACM, pp. 344–358.
[50] Yang, J., Sabnis, A., Berger, D. S., Rashmi, K., and Sitaraman, R. K. C2dn: How to harness erasure codes at the edge for efficient content delivery. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 1159–1177.
[51] Zhang, J., and Bareinboim, E. Transfer learning in multi-armed bandit: A causal approach. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (2017), pp. 1778–1780.
[52] Zhang, Q., Xiang, Z., Zhu, W., and Gao, L. Cost-based cache replacement and server selection for multimedia proxy across wireless internet. *IEEE Transactions on Multimedia 6*, 4 (2004), 587–598.

A APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A.1 Offline Clustering and Expert Sets

We assume that each offline-collected traffic trace contains sequences of requests indexed by a triple of the ID, size, and timestamp associated with the requested object.

Forming the clusters: For each trace, we compute a variety of statistics to use in clustering. In our setting, we found the following statistics to serve as useful features; we note Darwin allows the CDN server operators to use other features, too. The features are: *(a)*. Average request size, *(b)*. Vector of first n average inter-arrival times: The n^{th} inter-arrival time is the time elapsed between $n+1$ successive requests with the same ID, and *(c)*. Vector of first m average stack distances; the m^{th} stack distance measures the cumulative size of all requests received between $m+1$ successive requests with the same ID. The averages above are over all the choices of object IDs; n, m are hyperparameters. Together, these statistics summarize the trace and serve as features to cluster traces using the K-means clustering algorithm. The total number of clusters $N_{clusters}$ to be formed can also be tuned as necessary.

Clusters to experts sets: We evaluate the HOC hit rate of each expert over all the traces offline. For each trace, we collect experts that achieve hit rates within $\theta = 1\%$ of its best-performing expert to form the trace-level "best expert set". We then take the union of the trace-level best expert sets of all traces in a cluster to form the cluster-level best expert set.

This offline process results in a map from the features to a set of experts that are best suited for these features. There are two reasons that motivate the association process: *(a)*. The cluster-level best expert sets can potentially be much smaller than the total number of available experts, *(b)*. The $\theta = 1\%$ threshold above potentially captures the true best experts of the traces with similar features that were not present in the logs. Both these reasons will help accelerate learning in the online phase of the caching process.

A.2 Proofs of Theorems in Section 4.2

In our analysis, we assume that rewards are sampled from Gaussian distributions of known means, whereas in the experiments, the rewards (e.g., hit rates) are bounded over $[0, 1]$. The choice of Gaussian rewards is intentional because it leads to closed-form expressions and thus provides greater insight into the benefits of the algorithm (e.g., $O(1)$ with respect to K, the number of experts, as opposed to linear dependence without our approach). Our methods

apply with more generality (and can be numerically evaluated) and readers can replace the Gaussian-specific quantities we use with their respective counterparts and deploy the strategy we develop.

We present the proofs of our theoretical results and some discussions around the same here. Our analysis follows similar arguments to that of the Track and Stop algorithm of [19] for standard bandit feedback (We also refer to Chapter 33 in [24] for more discussions). Owing to our increased feedback through the cross-expert prediction networks and our modified estimators in Equation 1, we require novel concentrations inequalities to establish the soundness of Algorithm 1 as in Theorem 1. Recall that any triple (π, τ, ψ) of expert sequence selection policy, stopping time and expert recommendation rule (respectively) is said to be $\delta-$sound if $\mathbb{P}_{\mu,\Sigma,\pi}(\psi(\hat{\mu}_\tau) \neq i^*(\mu)) \leq \delta$.

We will first establish any-time concentration bounds on the estimators $\hat{\mu}_i(t)$ around their mean μ_i that do not depend on the policy used to design the expert selection sequence. These will instruct our design for the threshold function $\beta_t(\delta, \Sigma)$.

A.2.1 Concentration Bounds for Estimators in Equation 1.

Lemma 3. *Let $\sigma_{min}, \sigma_{max}, M$ and κ be as defined in Theorem 1. Then, for some constant C, we have that for any time t,*

$$\mathbb{P}\left(\sum_{i=1}^{K}(\hat{\mu}_i(t) - \mu_i)^2 \geq K\sigma_{max}^2 + \frac{KM^2}{2\kappa}\sqrt{\frac{\log(2/\delta)}{Ct}}\right) \leq \delta.$$

Proof: In order to prove these results, we will use a variant of McDiarmid's inequality with bad events as in [29]; specifically, we apply Theorem 7.8 of [15]. To this end, we define the event $\mathcal{E} = \{\forall t, \forall i, Y_i(t) \in [-M, M]\}$ ($\mathcal{E}^C$ is our 'bad event'). We assume $\mathcal{E}$ holds (and suppress the conditioning notation to ease presentation) until otherwise mentioned. We define the function

$$f(E_1, \{Y_i(1)\}, ..., E_t, \{Y_i(t)\}) = \sum_{i=1}^{K}\left(\frac{\sum_{n=1}^{t}\frac{Y_i(n)}{\sigma^2_{E_n,i}}}{\sum_{n=1}^{t}\frac{1}{\sigma^2_{E_n,i}}} - \mu_i\right)^2.$$

Note that the arguments of this function are random variables: $Y_i(t)$ is the reward obtained from expert i in round t and E_t is the deployed expert in this round. To apply Theorem 7.8 of [15], we need to show that the expected value of this function is bounded and that the function satisfies the bounded difference criteria therein. The former is guaranteed as since the expected value of the $f(\cdot)$ is only a function of $\{\mu_i\}_{i=1}^K$ and $\{\sigma_{j,i}\}_{i,j=1}^K$. By assumption, $\max_{i\in[K]}\mu_i < \infty$ and $\max_{i,j\in[K]}\sigma_{j,i} < \infty$ and thus, the expected value is finite. We are only left to set up the bounded differences appropriately. For this, we will treat the changes in E_n and those in $Y_i(t)$ separately.

Case 1: $Y_i(m) \to Y_i'(m)$

$$\begin{aligned}
&f(E_1, \{Y_i(1)\}, ..., Y_i(m), ..., E_t, \{Y_i(t)\}) \\
&\quad - f(E_1, \{Y_i(1)\}, ..., Y_i'(m), ..., E_t, \{Y_i(t)\}) \\
&= \left(\frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} + \frac{Y_i(m)}{\sigma^2_{E_m,i}}}{\rho_i(t)} - \mu_i\right)^2 \\
&\quad - \left(\frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} + \frac{Y_i'(m)}{\sigma^2_{E_n,i}}}{\rho_i(t)} - \mu_i\right)^2 \\
&= \left(\frac{Y_i(m)}{\sigma^2_{E_m,i}\rho_i(t)}\right)^2 - \left(\frac{Y_i'(m)}{\sigma^2_{E_m,i}\rho_i(t)}\right)^2 \\
&\quad + 2\cdot\left(\frac{Y_i(m)}{\sigma^2_{E_m,i}\rho_i(t)} - \frac{Y_i'(m)}{\sigma^2_{E_m,i}\rho_i(t)}\right)\cdot\left(\frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}}}{\rho_i(t)} - \mu_i\right)
\end{aligned}$$

Treating the two parts separately, we have that

$$\begin{aligned}
\left|\left(\frac{Y_i(m)}{\sigma^2_{E_m,i}\rho_i(t)}\right)^2 - \left(\frac{Y_i'(m)}{\sigma^2_{E_m,i}\rho_i(t)}\right)^2\right| &\leq \left|\frac{Y_i(m)^2 - Y_i'(m)^2}{\rho_i(t)^2\sigma^4_{E_m,i}}\right| \\
&\leq \frac{M^2\sigma^4_{max}}{t^2\sigma^4_{min}} = \frac{M^2}{t^2\kappa^2},
\end{aligned}$$

$$\begin{aligned}
&2\left|\frac{Y_i(m) - Y_i'(m)}{\sigma^2_{E_m,i}\rho_i(t)}\left(\frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}}}{\rho_i(t)} - \mu_i\right)\right| \\
&\quad \leq \frac{4M\sigma^2_{max}}{t\sigma^2_{min}}\left|\frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{A_n,i}}}{\rho_i(t)} - \mu_i\right| \\
&\quad \leq \frac{4M}{t\kappa}\left(\frac{M(t-1)}{t\kappa} + 1\right).
\end{aligned}$$

Combining these, we get

$$\begin{aligned}
&|f(E_1, \{Y_i(1)\}, ..., E_m, Y_1(m), ..., Y_i(m), ...Y_K(m), ..., E_t, \{Y_i(t)\}) \\
&- f(E_1, \{Y_i(1)\}, ..., E_m, Y_1(m), ..., Y_i'(m), ...Y_K(m), ..., E_t, \{Y_i(t)\})| \\
&\quad \leq \frac{4M(M-\kappa)}{t\kappa^2} + o\left(\frac{1}{t}\right) \qquad (4)
\end{aligned}$$

Case 2: $E_m \to E_m'$

$$\begin{aligned}
&f(E_1, \{Y_i(1)\}, ..., E_m, \{Y_i(m)\}, ..., E_t, \{Y_i(t)\}) \\
&\quad - f(E_1, \{Y_i(1)\}, ..., E_m', \{Y_i(m)\}, ..., E_t, \{Y_i(t)\}) \\
&\quad = \sum_{i=1}^{K}\left[\left(\frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} + \frac{Y_i(m)}{\sigma^2_{E_m,i}}}{\sum_{n\neq m}\sigma^{-2}_{E_n,i} + \sigma^{-2}_{E_m,i}} - \mu_i\right)^2\right. \\
&\qquad \left. - \left(\frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} + \frac{Y_i(m)}{\sigma^2_{E_m',i}}}{\sum_{n\neq m}\sigma^{-2}_{E_n,i} + \sigma^{-2}_{E_m',i}} - \mu_i\right)^2\right] \\
&\quad = \sum_{i=1}^{K}(A_i^2 - A_i'^2) + (B_i^2 + B_i'^2) \\
&\qquad - 2\mu(A_i + B_i - A_i' - B_i') + 2(A_iB_i - A_i'B_i')
\end{aligned}$$

where

$$A_i = \frac{1}{\rho_i(t)}\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}}, \quad A_i' = \frac{1}{\rho_i'(t)}\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}}$$

$$B_i = \frac{1}{\rho_i(t)}\frac{Y_i(m)}{\sigma^2_{E_m,i}}, \qquad B'_i = \frac{1}{\rho'_i(t)}\frac{Y_i(m)}{\sigma^2_{E'_m,i}}$$
$$\rho'_i(t) = \sum_{n\neq m} \sigma^{-2}_{E_n,i} + \sigma^{-2}_{E'_m,i}.$$

First, we have

$$|A_i^2 - A_i'^2| = \left(\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}}\right)^2 \cdot \left(\frac{1}{\rho_i(t)^2} - \frac{1}{\rho_i'^2(t)}\right)$$
$$\leq M^2C^2 \cdot \left|\frac{1}{(C+\sigma^{-2}_{E_m,i})^2} - \frac{1}{(C+\sigma^{-2}_{E'_m,i})^2}\right|$$

In the above, $C = \sum_{n\neq m}\sigma^{-2}_{E_n,i}$. We can bound the second term as:

$$\left|\frac{1}{(C+\sigma^{-2}_{E_m,i})^2} - \frac{1}{(C+\sigma^{-2}_{E'_m,i})^2}\right|$$
$$= \left|\frac{\sigma^4_{E_m,i}(1+C\sigma^2_{E'_m,i})^2 - \sigma^4_{E_m,i}(1+C\sigma^2_{E_m,i})^2}{(1+C\sigma^2_{E_m,i})^2(1+C\sigma^2_{E'_m,i})^2}\right|$$
$$\leq \frac{1}{(1+C\sigma^2_{min})^4}\left|\sigma^4_{E_m,i} - \sigma^4_{E'_m,i} + 2C\sigma^2_{E_m,i}\sigma^2_{E'_m,i}(\sigma^2_{E_m,i} - \sigma^2_{E'_m,i})\right|$$
$$= \frac{\sigma^2_{E_m,i} + \sigma^2_{E'_m,i} + 2C\sigma^2_{E_m,i}\sigma^2_{E'_m,i}}{(1+C\sigma^2_{min})^4}\left|\sigma^2_{E_m,i} - \sigma^2_{E'_m,i}\right|$$
$$\leq \frac{(\sigma^2_{max} - \sigma^2_{min})(\sigma^2_{min} + \sigma^2_{max} + 2C\sigma^2_{min}\sigma^2_{max})}{(1+C\sigma^2_{min})^4}$$

Therefore,

$$|A_i^2 - A_i'^2|$$
$$\leq \frac{M^2C^2(\sigma^2_{max} - \sigma^2_{min})(\sigma^2_{min} + \sigma^2_{max} + 2C\sigma^2_{min}\sigma^2_{max})}{(1+C\sigma^2_{min})^4}$$
$$\leq \frac{M^2(t-1)^2(1+\kappa+2(t-1)\kappa)(1-\kappa)}{(1+\kappa(t-1))^4}$$
$$= \frac{2M^2\kappa(1-\kappa)t^3}{(1+\kappa(t-1))^4} + o\left(\frac{1}{t}\right) \tag{5}$$

Next, we have

$$|B_i^2 - B_i'^2|$$
$$\leq \left|Y_i(m)\left(\frac{1}{\sigma^4_{E_m,i}(C+\sigma^{-2}_{E_m,i})^2} - \frac{1}{\sigma^4_{E'_m,i}(C+\sigma^{-2}_{E'_m,i})^2}\right)\right|$$
$$\leq M^2\left|\frac{1}{(1+\sigma^2_{E_m,i}C)^2} - \frac{1}{(1+\sigma^2_{E'_m,i}C)^2}\right|$$
$$\leq M^2\left|\frac{(\sigma^4_{E'_m,i} - \sigma^4_{E_m,i})C^2 + 2C(\sigma^2_{E'_m,i} - \sigma^2_{E_m,i})}{(1+\sigma^2_{E_m,i}C)^2(1+\sigma^2_{E'_m,i}C)^2}\right|$$
$$\leq \frac{M^2}{(1+\sigma^2_{min}C)^4}\cdot\left|(\sigma^4_{E'_m,i} - \sigma^4_{E_m,i})C^2 + 2C(\sigma^2_{E'_m,i} - \sigma^2_{E_m,i})\right|$$
$$\leq \frac{M^2(\sigma^2_{max} - \sigma^2_{min})\left((\sigma^2_{max} + \sigma_{min^2})C^2 + 2C\right)}{(1+\sigma^2_{min}C)^4}$$
$$\leq \frac{M^2(t-1)(\sigma^2_{max} - \sigma^2_{min})\left(2 + (\sigma^2_{min} + \sigma^2_{max})\frac{t-1}{\sigma^2_{max}}\right)}{\sigma_{max^2}(1+\kappa(t-1))^4}$$
$$= \frac{M^2(t-1)(1-\kappa)(2+(\kappa+1)(t-1))}{(1+\kappa(t-1))^4} = O\left(\frac{1}{t^2}\right) \tag{6}$$

Next, we consider

$$A_i + B_i - A'_i - B'_i = \frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} + \frac{Y_i(m)}{\sigma^2_{E_m,i}}}{\rho_i(t)} - \frac{\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} + \frac{Y_i(m)}{\sigma^2_{E'_m,i}}}{\rho'_i(t)}$$
$$= \frac{\alpha+\beta}{\gamma+\delta} - \frac{\alpha+\beta'}{\gamma+\delta'}$$

where

$$\alpha = \sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{A_E,i}}, \qquad \gamma = \sum_{n\neq m}\frac{1}{\sigma^2_{E_n,i}}$$
$$\beta = \frac{Y_i(m)}{\sigma^2_{E_m,i}}, \qquad \beta' = \frac{Y_i(m)}{\sigma'^2_{E_m,i}}$$
$$\delta = \frac{1}{\sigma^2_{E_m,i}}, \qquad \delta' = \frac{1}{\sigma'^2_{E_m,i}}.$$

Thus,

$$A_i + B_i - A'_i - B'_i$$
$$= \frac{\alpha(\delta'-\delta) + \gamma(\beta-\beta') + \beta\delta' - \beta'\delta}{(\gamma+\delta)(\gamma+\delta')}$$
$$= \left(\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} - Y_i(m)\sum_{n\neq m}\frac{1}{\sigma^2_{E_n,i}}\right) \times \left(\frac{\sigma^2_{E_m,i} - \sigma^2_{E'_m,i}}{\sigma^2_{E_m,i}\sigma^2_{E'_m,i}}\right)\cdot\frac{1}{(\gamma+\delta)(\gamma+\delta')}$$
$$= \left(\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} - Y_i(m)\sum_{n\neq m}\frac{1}{\sigma^2_{E_n,i}}\right) \times \frac{\sigma^2_{E_m,i} - \sigma^2_{E'_m,i}}{(1+C\sigma^2_{E_m,i})(1+C\sigma^2_{E'_m,i})}$$
$$\leq \left(\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}} - Y_i(m)\sum_{n\neq m}\frac{1}{\sigma^2_{E_n,i}}\right)\cdot\frac{\sigma^2_{E_m,i} - \sigma^2_{E'_m,i}}{(1+C\sigma^2_{min})^2}$$
$$\implies |A_i + B_i - A'_i - B'_i| \leq \frac{2MC(\sigma^2_{max} - \sigma^2_{min})}{(1+C\sigma^2_{min})^2}$$
$$\leq \frac{2M(t-1)(1-\kappa)}{(1+(t-1)\kappa)^2} = \frac{2Mt(1-\kappa)}{(1+(t-1)\kappa)^2} + o\left(\frac{1}{t}\right) \tag{7}$$

And finally,

$$|A_iB_i - A'_iB'_i|$$
$$= \left|\frac{Y_i(m)\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}}}{\sigma^2_{E_m,i}\rho_i^2(t)} - \frac{Y_i(m)\sum_{n\neq m}\frac{Y_i(n)}{\sigma^2_{E_n,i}}}{\sigma^2_{E'_m,i}\rho_i'^2(t)}\right|$$
$$\leq M^2C\left|\frac{1}{\sigma^2_{E_m,i}(C+\sigma^{-2}_{E_m,i})^2} - \frac{1}{\sigma^2_{E'_m,i}(C+\sigma^{-2}_{E'_m,i})^2}\right|$$

$$
\begin{aligned}
&= M^2C\left|\frac{\sigma^2_{E_m,i}}{(1+C\sigma^2_{E_m,i})^2}-\frac{\sigma^2_{E'_m,i}}{(1+C\sigma^2_{E'_m,i})^2}\right| \\
&= M^2C\left|\frac{\sigma^2_{E_m,i}-\sigma^2_{E'_m,i}+C^2\sigma^2_{E_m,i}\sigma^2_{E'_m,i}(\sigma^2_{E'_m,i}-\sigma^2_{E_m,i})}{\left((1+C\sigma^2_{E_m,i})(1+C\sigma^2_{E'_m,i})\right)^2}\right| \\
&\leq \frac{M^2C}{\left(1+C\sigma^2_{min}\right)^4} \\
&\quad\times\left|\sigma^2_{E_m,i}-\sigma^2_{E'_m,i}+C^2\sigma^2_{E_m,i}\sigma^2_{E'_m,i}(\sigma^2_{E'_m,i}-\sigma^2_{E_m,i})\right| \\
&\leq \frac{M^2C(\sigma^2_{max}-\sigma^2_{min})(C^2\sigma^2_{min}\sigma^2_{max}-1)}{\left(1+C\sigma^2_{min}\right)^4} \\
&\leq \frac{M^2(t-1)(1-\kappa)((t-1)^2\kappa-1)}{(1+(t-1)\kappa)^4} \\
&= \frac{M^2\kappa(1-\kappa)t^3}{(1+(t-1)\kappa)^4}+o\left(\frac{1}{t}\right) \qquad (8)
\end{aligned}
$$

Combinging Equations (5), (6), (7) and (8), we get:

$$
\begin{aligned}
&|f(E_1,\{Y_i(1)\},...,E_m,\{Y_i(m)\},...,E_t,\{Y_i(t)\}) \\
&\quad f(E_1,\{Y_i(1)\},...,E'_m,\{Y_i(m)\},...,E_t,\{Y_i(t)\})| \\
&\quad\leq K\left(\frac{3\kappa(1-\kappa)t^3}{(1+(t-1)\kappa)^4}+\frac{4Mt(1-\kappa)}{(1+(t-1)\kappa)^2}\right)+o\left(\frac{1}{t}\right) \\
&\leq \frac{MK\kappa(1-\kappa)(3M+4\kappa)t^3}{(1+(t-1)\kappa)^4}+o\left(\frac{1}{t}\right) \\
&\leq \frac{MK(1-\kappa)(3M+4\kappa)}{16\kappa^3t}+o\left(\frac{1}{t}\right). \qquad (9)
\end{aligned}
$$

Therefore, we can write that the function $f(\cdot)$ follows the bounded difference condition of Theorem 7.8 in [15] using Equations 4 and 9 with

$$
c_i = \begin{cases} \frac{C_1KM(1-\kappa)(3M+4\kappa)}{16\kappa^3t} & \text{; for changes in expert choices} \\ \frac{C_2M(M-\kappa)}{t\kappa^2} & \text{; for changes in reward samples} \end{cases}
$$

Note that there are t random variables of the first type and Kt random variables of the second. This leads to

$$
\begin{aligned}
\sum_{i=1}^{(K+1)t} c_i^2 &= t\left(\frac{C_1KM(1-\kappa)(3M+4\kappa)}{16\kappa^3t}\right)^2 \\
&\quad + Kt\left(\frac{C_2M(M-\kappa)}{t\kappa^2}\right)^2 \\
&\leq O\left(\frac{K^2M^4}{\kappa^2t}\right)
\end{aligned}
$$

Applying McDiarmid's inequality with C large, we get

$$
\begin{aligned}
&\mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\epsilon\right) \\
&\quad\leq \mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq \mathbb{E}\left[\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\right]+\epsilon\right) \\
&\quad\leq \exp\left(-\frac{2\epsilon^2\kappa^2t}{CK^2M^4}\right)
\end{aligned}
$$

We can re-write this as

$$
\begin{aligned}
&\mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\epsilon\right)\leq\exp\left(-\frac{2\epsilon^2\kappa^2t}{C_3K^2M^4}\right) \\
&\iff \mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\tfrac{KM^2}{2\kappa}\sqrt{\tfrac{\log(2/\delta)}{C_3t}}\right)\leq\frac{\delta}{2}.
\end{aligned}
$$

Recall that all arguments hitherto were under the event $\mathcal{E}=\{\forall t,\forall i,Y_i(t)\in[-M,M]\}$ with $\mathbb{P}(\mathcal{E}^C)\leq\frac{\delta}{2}$ by definition of M. Thus, we get:

$$
\begin{aligned}
&\mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\tfrac{KM^2}{2\kappa}\sqrt{\tfrac{\log(2/\delta)}{C_3t}}\right) \\
&=\mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\tfrac{KM^2}{2\kappa}\sqrt{\tfrac{\log(2/\delta)}{C_3t}},\mathcal{E}\right) \\
&\quad+\mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\tfrac{KM^2}{2\kappa}\sqrt{\tfrac{\log(2/\delta)}{C_3t}},\mathcal{E}^C\right) \\
&\leq\mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\tfrac{KM^2}{2\kappa}\sqrt{\tfrac{\log(2/\delta)}{C_3t}}\middle|\mathcal{E}\right) \\
&\quad+\mathbb{P}(\mathcal{E}^C) \\
&\leq\delta.
\end{aligned}
$$

This completes the proof. ∎

A.2.2 Proof of Theorem 1. We are now ready to prove our soundness result. The analysis here is similar to that in Proposition 12 of [19] and Lemma 33.7 in [24]. The key difference being that we leverage Lemma 3 in place of standard concentrations on empirical estimators.

Proof: [Proof of Theorem 1] By definition of τ, Z_t, we have that

$$
\{\nu\in\mathcal{E}_{alt}(\hat{\nu})\}\subseteq\left\{\sum_{i=1}^KT_i(\tau)\sum_{j=1}^K\frac{(\hat{\mu}_j(t)-\mu_i)^2}{2\sigma^2_{i,j}}\geq\beta_\tau(\delta,\Sigma)\right\}
$$

Without loss of generality, let $i^*(\nu)=1$. Thus, we have

$$
\begin{aligned}
&\mathbb{P}(1\neq\psi(\hat{\nu}(\tau))) \\
&\leq\mathbb{P}(\nu\in\mathcal{E}_{alt}(\hat{\nu}(\tau))) \\
&\leq\mathbb{P}\left(\sum_{i=1}^KT_i(\tau)\sum_{j=1}^K\frac{(\hat{\mu}_j(\tau)-\mu_j)^2}{2\sigma^2_{i,j}}\geq\beta_\tau(\delta,\Sigma)\right) \\
&\leq\mathbb{P}\left(\sum_{j=1}^K\frac{(\hat{\mu}_j(\tau)-\mu_j)^2}{2}\sum_{i=1}^K\frac{T_i(\tau)}{\sigma^2_{i,j}}\geq\beta_\tau(\delta,\Sigma)\right) \\
&\leq\mathbb{P}\left(\frac{\tau}{2\sigma^2_{min}}\sum_{j=1}^K(\hat{\mu}_j(\tau)-\mu_j)^2\geq\beta_\tau(\delta,\Sigma)\right) \\
&\leq\mathbb{P}\left(\sum_{i=1}^K(\hat{\mu}_i(t)-\mu_i)^2\geq K\sigma^2_{max}+\frac{KM^2}{2\kappa}\sqrt{\frac{\log(1/\delta)}{C\tau}}\right) \\
&\leq\delta.
\end{aligned}
$$

Where the final inequality follows from Lemma 3. ∎

A.2.3 Proof of Theorem 2. We now move on to the proof of our stopping time scaling. This follows the sequence of arguments used in proving Theorem 33.6 in [24]. The key differences here are due to the modified definition of $\Phi(\nu, \alpha)$ in Equation (2). Further, the lack of explicit exploration in Algorithm 1 also changes how we bound the average stopping time.

Proof:[Proof of Theorem 2] For an environment ν, we have

$$\Phi(\nu,\alpha) = \inf_{\nu' \in \mathcal{E}_{alt}(\nu)} \sum_{i=1}^{K} \alpha_i \sum_{j=1}^{K} \frac{(\mu_j - \mu'_j)^2}{2\sigma_{ij}^2}$$
$$= \frac{1}{2} \min_{k \neq k^*} \frac{w_{k^*} \cdot w_k \cdot \Delta_k^2}{w_{k^*} + w_k} := \frac{1}{2} \min_{k \neq k^*} f_k(w_{k^*}, w_k)$$

Here, we have $k^* = i^*(\nu)$ and $w_k = \sum_{i=1}^{K} \frac{\alpha_i}{\sigma_{ik}^2}$. Note that we have $Z_t = \Phi(\hat{\nu}_t, T_i(t))$. We also define

$$\alpha^*(\nu) = \operatorname*{argmax}_{\alpha \in \mathcal{P}_{K-1}} \Phi(\nu, \alpha), \quad \frac{1}{c^*(\nu)} = \Phi(\nu, \alpha^*(\nu)).$$

We now study the quantity $\alpha^*(\nu)$, the optimal proportion of plays for environment ν. Since w_k are affine in α_i and $\Phi(\nu, \cdot)$ is a minimum of concave functions in w_i, $\Phi(\nu, \cdot)$ is thus concave in its second argument. $\alpha^*(\nu)$ are to be chosen such that the corresponding w_k^* equalize the functions $f_k(w_{k^*}^*, w_k^*)$ for all $k \neq k^*$ to a constant. Therefore, letting $\Phi(\nu) = \frac{w_{k^*}^* w_k^* \Delta_k^2}{2(w_{k^*}^* + w_k^*)}$, we have that the optimal $\alpha^*(\nu)$ are solutions to the system of equations given by

$$\sum_{i=1}^{K} \alpha_i^*(\nu) = 1, \quad w_k^* = \sum_{i=1}^{K} \frac{\alpha_i^*(\nu)}{\sigma_{ik}^2}, \quad w_k^* = \frac{2 w_{k^*}^* \Phi(\nu)}{\Delta_k^2 w_{k^*}^* - 2\Phi(\nu)}.$$

The solutions are roots of some polynomial, and thus are finite in number. Further, due to concavity of $\Phi(\nu, \cdot)$, there are either infinitely many maxima or just one. Combining these two facts we get that $\alpha^*(\nu)$ is unique.

We define the metric $d(\nu_1, \nu_2) = \|\mu(\nu_1) - \mu(\nu_2)\|_\infty$ and note that under this metric on the space of environments, the function $\alpha^*(\nu)$ is continuous at every ν.

We now establish the finiteness of the mean of 3 random times that will help in establishing our results.

1. Consider the random time $\tau_\nu(\epsilon) = 1 + \max\{t : d(\hat{\nu}_t, \nu) \geq \epsilon\}$. We show that $\mathbb{E}[\tau_\nu(\epsilon)] < \infty$ for any $\epsilon > 0$. For this, we define the following random variable:

$$\Lambda = \max\left\{\lambda \geq 1 : \forall t, d(\hat{\nu}_t, \nu) \leq \sqrt{\frac{2\sigma_{max}^2 \log(\lambda K t(t+1))}{t}}\right\}$$

Since all rewards have variance upper bounded by σ_{max}^2, union bounding and Gaussian concentrations give that $\mathbb{P}(\Lambda \geq x) \leq \frac{1}{x}$. Additionally, we have

$$\mathbb{E}[\log^2 \Lambda] = \int_0^\infty \mathbb{P}(\lambda \geq \exp(\sqrt{x}))dx$$
$$\leq \int_0^\infty \exp(-\sqrt{x})dx = 2.$$

Therefore, by definition, we have

$$\tau_\nu(\epsilon) \leq 1 + \max\left\{t : \sqrt{\frac{2\sigma_{max}^2 \log(\Lambda K t(t+1))}{t}} > \epsilon\right\}$$
$$\implies \mathbb{E}[\tau_\nu(\epsilon)] \leq O(\mathbb{E}[\log^2 \Lambda]) = O(1).$$

2. Consider the random time $\tau_\alpha(\epsilon) = 1 + \max\{t : \|\alpha^*(\hat{\nu}_t) - \alpha^*(\nu)\|_\infty \geq \epsilon\}$. Let $w(\epsilon) = \inf\{x : d(w, \nu) \leq x \implies \|\alpha^*(w) - \alpha^*(\nu)\|_\infty \leq \epsilon\}$. By continuity of α^* satisfies that $\forall \epsilon > 0, w(\epsilon) > 0$. Therefore, $\mathbb{E}[\tau_\alpha(\epsilon)] \leq \mathbb{E}[\tau_\nu(w(\epsilon))] < \infty$.

3. Define the random time $\tau_T(\epsilon) = 1 + \max\{t : \|T(t)/t - \alpha^*(\nu)\|_\infty \geq \epsilon\}$, where $T(t) = \{T_1(t), ..., T_K(t)\}$ is the vector of the number of arm plays. Using the definition E_t, for $t \geq \frac{2K\tau_\alpha(\epsilon/2K)}{\epsilon}$, we have

$$T_i(t) \leq \max\{T_i(\tau_\alpha(\epsilon/2K)), 1 + t(\alpha_i^*(\nu) + \epsilon/2K)\}$$
$$\leq t\left(\alpha_i^*(\nu) + \frac{\epsilon}{K}\right)$$
$$\implies \frac{T_i(t)}{t} - \alpha_i^*(\nu) \leq \frac{\epsilon}{K}.$$

Therefore, $\mathbb{E}[\tau_T(\epsilon)] \leq \mathbb{E}\left[\frac{2K\tau_\alpha(\epsilon/2K)}{\epsilon}\right] < \infty$ for any $\epsilon > 0$ due to finiteness of $\mathbb{E}[\tau_\alpha(\epsilon)]$.

Finally, we now define

$$\tau_\beta(\epsilon) = 1 + \max\{t : t\Phi(\nu, \alpha^*(\nu)) < \beta_t(\delta) + \epsilon t\},$$
$$u(\epsilon) = \sup_{\omega,\alpha}\{\Phi(\omega, \alpha) : d(\omega, \nu) \leq \epsilon, \|\alpha - \alpha^*(\nu)\|_\infty \leq \epsilon\}$$

For any $t \geq \max\{\tau_n u(\epsilon), \tau_T(\epsilon), \tau_\beta(u(\epsilon))\}$, we have

$$tZ_t = t\Phi(\hat{\nu}_t, T(t)/T)$$
$$\geq t\left(\phi(\nu, \alpha^*(\nu)) - u(\epsilon)\right)$$
$$\geq \beta_t(\delta)$$
$$\implies \tau \leq \max\{\tau_\nu(\epsilon), \tau_T(\epsilon), \tau_\beta(u(\epsilon))\}$$
$$\implies \mathbb{E}[\tau] \leq \mathbb{E}[\tau_\nu(\epsilon)] + \mathbb{E}[\tau_T(\epsilon)] + \mathbb{E}[\tau_\beta(u(\epsilon)]$$
$$\implies \limsup_{\delta \to 0} \frac{\mathbb{E}[\tau]}{\log(1/\delta)} \leq \limsup_{\delta \to 0} \frac{\tau_\beta(u(\epsilon))}{\log(1/\delta)}$$

Where the final inequality follows since $\tau_\nu(\epsilon), \tau_T(\epsilon)$ do not depend on δ and $\tau_\beta(\epsilon)$ is deterministic.

Finally, with $C_1 = \frac{K}{2\kappa}, C_2 = \frac{M^2 K}{2\sigma_{min}^2 \kappa}$, we have the chain

$$t\Phi(\nu, \alpha^*(\nu)) < \beta_t(\delta) + u(\epsilon)t$$
$$\implies t\left(\Phi(\nu, \alpha^*(\nu)) - u(\epsilon)\right) < t \cdot C_1 + C_2 \cdot \sqrt{t \log(1/\delta)}$$
$$\implies \frac{t}{\log(1/\delta)} < \left(\frac{C_2}{\Phi(\nu, \alpha^*(\nu)) - u(\epsilon) - C_1}\right)^2$$

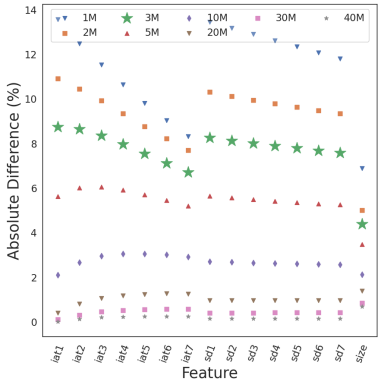
Taking $\epsilon \to 0$, we have $u(\epsilon) \to 0$ and thus,

$$\lim_{\delta \to \infty} \frac{\mathbb{E}[\tau]}{\log(1/\delta)} = \left(\frac{C_2}{\Phi(\nu, \alpha^*(\nu)) - C_1}\right)^2.$$

The result follows by substituting the values of C_1 and C_2. ∎

Baseline	Avg.Improvement Rate
f2s10	15.67
f2s20	4.84
f2s50	10.59
f2s100	19.21
f2s500	25.39
f2s1000	26.49
f3s10	22.10
f3s20	9.03
f3s50	7.17
f3s100	12.00
f3s500	17.05
f3s1000	18.09
f4s10	20.75
f4s20	14.76
f4s50	10.04
f4s100	12.10
f4s500	15.75
f4s1000	16.63
f5s10	34.45
f5s20	19.87
f5s50	13.78
f5s100	14.46
f5s500	16.94
f5s1000	17.68
f6s10	39.12
f6s20	24.28
f6s50	17.31
f6s100	17.21
f6s500	19.10
f6s1000	19.72
f7s10	43.07
f7s20	28.16
f7s50	20.65
f7s100	20.13
f7s500	21.63
f7s1000	22.20
Percentile	18.91
HillClimbing-$\Delta_s = 10$	8.00
HillClimbing-$\Delta_s = 20$	3.00
DirectMapping	7.21
AdaptSize	19.96

Table 2: "A Comparison of Average Improvement Rate of Darwin Relative to Baselines"

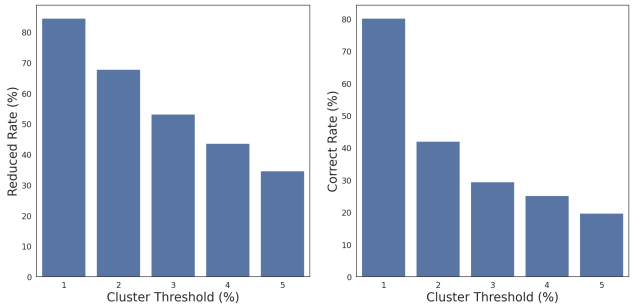


(a) 100M Request Traces

Figure 8: Convergence of features over 100M-length online test traces: Corresponding plot to Figure 5a. Empirical features are within 10% of their true values using $N_{warm-up} = 3M$.

A.3 Additional Results from Section 6

We present additional results from our evaluations.



(a) Expert Reduced Rate **(b) Expert within θ%**

Figure 9: Expert number reduction after clustering: Figure 9a presents the average reduction % of experts for different values of cluster threshold θ. Figure 9b displays the average fraction of experts in the expert sets within θ% of each other after clustering.

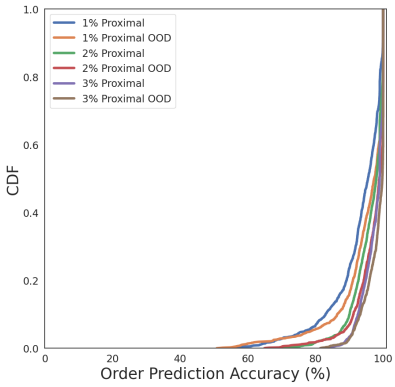


Figure 10: Out of Distribution performance of cross-expert neural network predictors, in addition to in-distribution performance: Corresponding figure to Figure 5c

We summarize the full set of average rate of improvement results comparing Darwin against baselines (other static experts, various configurations of HillClimbing, Percentile, AdaptSize and Directmapping) in Table 2.

Figure 8a confirms our claim that empirical features converge to within a 10% threshold with just the first $3M$ requests.

Figure 9 presents additional data on how offline clustering and expert set formation aids in reducing the number of experts under consideration.

Figure 10 illustrates the generalization performance of the cross-expert prediction networks by testing on mixtures that were not trained on.

Figure 11 shows the result for the reduction in the number of experts when we consider a third dimension of recency in addition to frequency and size. The original expert set size is 36 (6 frequencies, 2 sizes, 3 recencies) .

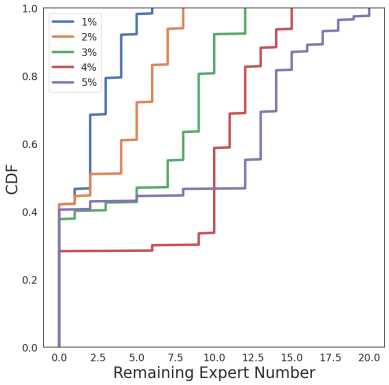


Figure 11: Expert reduction number after cluster experts, where experts use three knobs