Go-with-the-Winner: Performance Based Client-Side Server Selection

Chang Liu[†], Ramesh K. Sitaraman[†‡], and Don Towsley[†]
[†]University of Massachusetts, Amherst [‡]Akamai Technologies Inc.
{cliu, ramesh, towsley}@cs.umass.edu

***Abstract*—Content delivery networks deliver much of the world's web and video content by deploying a large distributed network of servers. We model and analyze a simple paradigm for client-side server selection that is commonly used in practice where each user independently measures the performance of a set of candidate servers and selects the one that performs the best. For web (resp. video) delivery, we propose and analyze a simple algorithm where each user randomly chooses two or more candidate servers and selects the server that provides the best hitrate (resp. bitrate). We prove that the algorithm converges quickly to an optimal state where all users receive the best hitrate (resp. bitrate), with high probability. We also show that if each user chooses just one random server instead of two, some users receive a hitrate (resp. bitrate) that tends to zero. We simulate our algorithm and evaluate its performance with varying choices of parameters, system load, and content popularity.**

I. Introduction

Modern content delivery networks (CDNs) host and deliver a large fraction of the world's web content, video content, and application services on behalf of enterprises that include most major web portals, media outlets, social networks, application providers, and news channels [17]. CDNs deploy large numbers of servers around the world that can store content and deliver that content to users who request it. When a user requests a content item, say a web page or a video, the user is directed to one of the CDN's servers that can serve the desired content to the user. The goal of a CDN is to maximize the performance perceived by the user while efficiently managing its server resources.

A key function of a CDN is *server selection* by which client software running on the user's computer or device, such as media player or a browser, is directed to a suitable server of a CDN [6]. The desired outcome of server selection is that each user is directed to a server that will provide the requested content with good performance. The performance metrics that are optimized vary by the content type. For instance, good performance for a user accessing a web page might mean low latency web page downloads. Good performance for a user watching a video might mean high bitrate video delivery by the server while avoiding video freezing and rebuffering [11].

Server selection can be performed in two distinct ways that are not mutually exclusive. *Network-side server selection* algorithms monitor the real-time characteristics of the CDN and the Internet. Such algorithms are often complex and measure liveness and CDN server load, as well as latency, loss, and bandwidth of the communication paths between servers and users. Using this information, the algorithm computes a good "mapping" of users to servers, such that each user is assigned a "proximal" server capable of serving that user's content [17]. This mapping is computed periodically and is typically made available to the client using the domain name system (DNS). Specifically, the user's browser or media player looks up the domain name of the content that it wants to download and receives as translation the IP address of the selected server.

A complementary approach to network-side server selection that is commonly is used is *client-side server selection* where the client embodies a server selection algorithm. The client software is typically unaware of the global state of the server infrastructure, the Internet, or other clients. Rather, the client software typically makes future server selection decisions based on its own historical performance measurements from past server downloads. Client-side server selection can often be implemented as a plug-in within media players, web browsers, and web download managers [2].

While client-side server selection can be used to select servers within a single CDN, it can also be used in a multi-CDN setting. Large content providers often make the same content available to the user via multiple CDNs. In this case, the client tries out the different CDNs and chooses the "best" server from across multiple CDNs. For instance, NetFlix uses three different CDNs and the media player incorporates a client-side server selection algorithm to choose the "best" server (and the corresponding CDN) using performance metrics such as achievable video bitrates [1]. Note also that in the typical multi-CDN case, both network-side and client-side server selection can be used together, where the former is used to choose the candidate servers from each CDN and the latter is used by the user to pick the "best" among all the candidates.

A. The Go-With-The-Winner paradigm

A common and intuitive paradigm that is often used for client-side server selection in practice is what we call *"Go-With-The-Winner"* that consists of an initial *trial period* during which each user independently "tries out" a set of *candidate servers* by requesting content or services from them (cf. Figure 1). Subsequently, each user independently *decides* on the "best" performing server using historical performance information that the user collected for the candidate servers during the trial period. It is commonly implemented in the

ISBN 978-3-901882-83-8 © 2016 IFIP

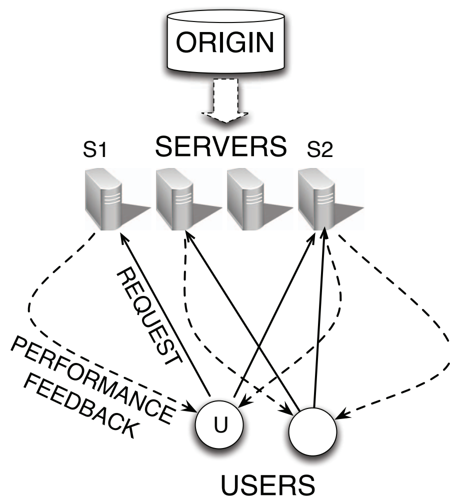


Fig. 1: Client-side Server Selection with the Go-With-The-Winner paradigm. User U makes request to two candidate servers $S1$ and $S2$. After a trial period of observing the performance provided by the candidate, the user selects the better performing server.

content delivery context that incorporate selecting a web or video content server from among a cluster of such servers.

Besides content delivery, the Go-With-The-Winner paradigm is also used for other Internet services, though we do not explicitly study such services in our work. For instance, BIND, which is the most widely deployed DNS resolver (i.e., DNS client) on the Internet, tracks performance as a smoothed value of historical round trip times (called SRTT) from past queries for a set of candidate name servers. BIND then chooses a particular name server to query in part based on the computed SRTT values [12]. It is also notable that BIND implementations incorporate randomness in the candidate selection process.

The three key characteristics of the Go-With-The-Winner paradigm are as follows.

1) *Distributed control.* Each user makes decisions in a distributed fashion using only knowledge available to it. There is no explicit information about the global state of the servers or other users, beyond what the user can infer from it's own historical experience.
2) *Performance feedback only.* There is no explicit feedback from a server to a user who requested service beyond what can be inferred by the performance experienced by the user.
3) *Choosing the "best" performer.* The selection criteria is based on historical performance measured by the user and consists of selecting the best server according to some performance metric (i.e., go with the winner).

Besides its inherent simplicity and naturalness, the paradigm is sometimes the only feasible and robust solution. For instance, in many settings, the client has no detailed knowledge of the state of the server infrastructure as it is managed and owned by other business entities. In this case, the primary feedback mechanism for the client is its own historical performance measurements.

While client-side server selection is widely implemented, its theoretical foundations are not well understood. A goal of our work is to provide such a foundation in the context of web and video content delivery. *It is not our intention to model a real-life client-side server selection process in its entirety which can involve other adhoc implemention-specific considerations. But rather we abstract an analytical model that we can explore to extract basic principles of the paradigm that are applicable in a broad context.*

B. Our contributions

We propose a simple theoretical model for the study of client-side server selection algorithms that use the Go-With-The-Winner paradigm. Using our model, we answer foundational questions such as how does randomness help in the trial period when selecting candidate servers? How many candidate servers should be selected in the trial phase? How long does it take for users to narrow down their choice and decide on a single server? Under what conditions does the selection algorithm converge to a state where all users have made correct server choices, i.e., selected servers provide good performance to their users? Some of our key results that help answer these questions follow.

(1) In Section II, in the context of web content delivery, we analyze a simple algorithm called GoWithTheWinner where each user independently selects two or more random servers as candidates and decides on the server that provides the best cache hit rate. We show that with high probability, the algorithm converges quickly to a state where no cache is overloaded and all users obtain a 100% hit rate. Furthermore, we show that two or more random choices of candidate servers are necessary, as just one random choice will result in some users (and some servers) incurring cache hit rates that tend to zero, as the number of users and servers tend to infinity. This work represents the first demonstration of the "power of two choices" phenomena in the context of client-side server selection for content delivery, akin to similar phenomena observed in balls-into-bins games [14], load balancing, circuit-switching algorithms [4], relay allocation for services like Skype [16], and multi-path communication [10].

(2) In Section III, in the context of video content delivery, we propose a simple algorithm called MaxBitRate where each user independently selects two or more random servers as candidates and decides on the server that provides the best bitrate for the video stream, We show that with high probability, the algorithm converges quickly to a state where no server is overloaded and all users obtain the required bitrate for their video to play without freezes. Further, we show that two or more random choices of candidate servers are necessary, as just one random choice will result in some users receiving bitrates that tend to zero, as the number of users and servers tends to infinity.

(3) In Section IV, we go beyond our theoretical model and simulate algorithm GoWithTheWinner in more complex settings. We establish an inverse relationship between the length of the history used for hitrate computation (denoted by τ) and the failure rate defined as the probability that the system converges to a non-optimal state. We show that as τ increases the convergence time increases, but the failure rate decreases. We also empirically evaluate the impact of the number of choices of candidate servers. We show that two or more random choices are required for all users to receive a 100% hitrate. Though even if only 70% of the users make two choices, it is sufficient for 95% of the users to receive a 100% hitrate. Finally, we show that the convergence time increases with system load. But, convergence time decreases when the exponent of power law distribution that describes content popularity increases.

II. Hit Rate Maximization for Web Content

The key measure of web performance is *download time* which is the time taken for a user to download a web object, such as an html page or an embedded image. CDNs enhance web performance by deploying a large number of servers in access networks "close" to the users. Each server has a cache capable of storing web objects. When a user requests an object, such as a web page, the user is directed to a server that can serve the object (cf. Figure 1). If the server already has the object in its cache, i.e, the user's request is a *cache hit*, the object is served from the cache to the user. In this case, the user experiences good performance, since the CDN's servers are proximal to the user and the object is downloaded quickly. However, if the requested object is not in the server's cache, i.e., the user's request is a *cache miss*, then the server first fetches it from the origin, places it in its cache, and then serves the object to the user. In the case of a cache miss, the performance experienced by the user is often poor since the origin server is typically far away from the server and the user. In fact, if there is a cache miss, the user would have been better off not using the CDN at all, since downloading the content directly from the content provider's origin would likely have been faster! Since the size of a server's cache is bounded, cache misses are inevitable. A key goal of server selection for web content delivery is to jointly orchestrate server assignment and content placement in caches such that the cache hit rate is maximized. While server selection in CDNs is a complex process [17], we analytically model the key elements that relate to content placement and cache hit rates, leaving other factors that impact performance such as server-to-user latency for future work.

A. Problem Formulation

Let U be a set of n_u users who each requests an object picked independently from a set C of size n_c using a popularity distribution $\{p_1, p_2, \ldots, p_{n_c}\}$, where the k-th most popular object in C is picked with probability p_k. The user then makes a sequence of requests for that content item to the set of available servers. In practice, users tend to stay with one website for a while, say reading the news or looking at a friend's posts. We model the sequence of requests generated by each user as a Poisson process with homogeneous arrival rate λ. Note that each request from user u can be sent to one or more servers selected from $S_u \subseteq S$, where S_u is the set of candidate servers for user u.

Let S be the set of n_s servers that are capable of serving content to the users. Each server can cache at most κ objects and a cache replacement policy such as LRU is used to evict objects when the cache is full. Given that the download time of a web object is significantly different when the request is a cache hit versus a cache miss, we make the assumption that the user can reliably infer if its request to download an object from a server resulted in a cache hit or a cache miss immediately after the download completes.

The objective of client-side server selection is for each user $u \in U$ to independently select a server $s \in S$ using only the performance feedback obtained on whether each request was a hit or a miss. Let the hit rate function $H(u, s, t)$ denote the probability of user u receiving a hit from server $s \in S_u$ at time t. We define the system-wide performance measure $H(t)$, as the best hit rate obtained by the worst user at time t,

$$H(t) \triangleq \min_{u \in U} \max_{s \in S_u} H(u, s, t), \tag{1}$$

a.k.a. the *minmax hit rate*. Our goal is to maximize $H(t)$. In the rest of the section, we describe a simple "Go-With-The-Winner" algorithm for server selection and show that it converges quickly to an optimal state, with high probability.

Note: Our formulation is intentionally simple so that it can model a variety of other situations in web content delivery. For instance, a single server could in fact model a cluster of front-end servers that share a single backend object cache. A single object can model a bucket of objects that cached together as is often done in a CDN context [17].

B. The GoWithTheWinner Algorithm

After each user $u \in U$ selects a content item and a set of σ servers S_u, the user executes algorithm *GoWithTheWinner* to select a server likely to always have the content. In this algorithm, each user locally executes a simple "Go-With-The-Winner" strategy of trying out σ randomly chosen candidate servers initially. For each server $s \in S_u$, the user keeps track of the most recent request results in a vector $\mathbf{h}^s = (h_1^s, h_2^s, \cdots, h_\tau^s)$ where $h_k^s = 1$ corresponds to the k-th recent request resulting in a hit from server s and $h_k^s = 0$ if otherwise. τ is the "sliding window size". Using the hit rates, each user then independently either chooses to continue with all the servers in S_u or decides on a single server that provided good performance. If there are multiple servers providing 100% hit rate, the user decides to use the first one found.

C. Analysis of Algorithm GoWithTheWinner

Here we analyze the case where $n_u = n_c = n_s = n$ and experimentally explore other variants where n_c and n_u are larger than n_s in Section II-D and IV. Let $H(t)$ be as defined in (1). If $\sigma \geq 2$, we show that with high probability $H(t) = 100\%$,

Algorithm 1: GoWithTheWinner

```
1  The current user u chooses a set of σ candidate servers
   S_u ⊆ S uniformly at random from all the servers;
2  for each s ∈ S_u do
3  |  set h^s ← (h^s_1, h^s_2, ··· , h^s_τ) = 0;
4  end
5  for each arrival of request do
6  |  set t to the current time;
7  |  Request content a_u from all servers s ∈ S_u;
8  |  for each server s ∈ S_u do
9  |  |  h^s_i ← h^s_{i−1}, 2 ≤ i ≤ τ;
10 |  |  h^s_1 ← if hit; h^s_1 ← 0, if miss;
11 |  |  compute hit rate H_τ(u, s, t) ← (Σ_{i=1}^τ h^s_i)/τ ;
12 |  |  if H_τ(u, s, t) = 100% then
13 |  |  |  decide on server s by setting S_u ← {s};
14 |  |  |  return;
15 |  |  end
16 |  end
17 end
```

for all $t \geq T$, where $T = O(\frac{\kappa}{\log(\kappa+1)}(\log n)^{\kappa+1}\log\log n)$. That is, the algorithm converges quickly with high probability to an optimal state where *every* user has decided on a single server that provides a 100% hit rate, and *every* server has the content requested by its users.

Definitions. A server s is said to be *overbooked* at some time t if users request more than κ distinct content items from server s, where κ is the number of content items a server can hold. Note that a server may have more than κ users and not be overbooked, provided the users collectively request a set of κ or fewer content items. Also, note that a server that is overbooked at time t is overbooked at every $t' \leq t$ since the number of users requesting a server can only remain the same or decrease with time. Finally, a user u is said to be *undecided* at time t if $|S_u| > 1$ and is said to be *decided* if it has settled on a single server to serve its content and $|S_u| = 1$. Note that each user starts out undecided at time zero, then decides on a server at some time t and remains decided in all future time later than t. Users calculate the hit rates of each of the available servers based on a history record of the last τ requests, where τ is called the sliding window size.

Lemma 1: If the sliding window size $\tau = \Theta(\log^{\kappa+1} n)$, the probability that some user $u \in U$ decides on an overbooked server $s \in S_u$ upon any request arrival is at most $1/n^{\Omega(1)}$.

Proof: If user u decides on server s then the current request together with the previous $\tau - 1$ requests are all hits. Let H_k, $k = 1, 2, \cdots, \tau$ be Bernoulli random variables, s.t. $H_k = 1$ if the most recent k-th request of u is a hit and $H_k = 0$ if it is a miss. To prove Lemma 1 we need to show

$$\mathbb{P}\left(\cap_{k=1}^{\tau}(H_k = 1)\right) \leq n^{-\Omega(1)}. \quad (2)$$

Let t_1 denote the time of the most recent request for content a_u from user u appears at server s, resulting in feedback H_1 to the user. Let $t_1 - \Delta$ be the time that the previous request for a_u arrives at s. Let $A_s = \{a_1, a_2, \cdots, a_M\}$ be the set of different content items requested at s, where $M > \kappa$. Let $N_i \geq 1$ be the number of users requesting a_i from s. WLOG, let $a_1 = a_u$ be the content that u requests, such that N_1 is the number of users requesting for a_u. Because we assume all the users generates requests with a Poisson process with arrival rate λ, the aggregated arrival rate of requests for a_u is then $N_1\lambda$. Thus Δ is an exponential random variable, $\Delta \sim Exp(N_1\lambda)$. Now we look at the number of different requests arrives between time $t_1 - \Delta$ and t_1. Let $X_i, i = 2, 3, \cdots, M$ be an indicator that a request for a_i arrives at server s during the time interval $(t_1 - \Delta, t_1)$, we have $X_i \sim Bernoulli(1 - e^{-N_i\lambda\Delta})$. Furthermore, let random variable $Y = \sum_{i=2}^{M} X_i$ be the number of different requests arrived in the time interval. With the server running on LRU replacement policy,

$$\mathbb{P}(H_1 = 0) = \mathbb{P}(Y \geq \kappa), \quad (3)$$

because for content a_u to be swapped out of the server, more than κ different requests other than that for a_u must have arrived. Equation (3) shows that H_1 only depends on the number different requests arrived after the previous request for a_u, which means events $H_k, k = 1, 2, \cdots, \tau$ are mutually independent.

Furthermore[1], because $N_i \geq 1$, we have $X_i \geq_d X'$ where $X' \sim Bernoulli(1 - e^{-\lambda\Delta})$. Thus,

$$Y = \sum_{i=2}^{M} X_i \geq_d \sum_{i=2}^{M} X' = Z,$$

where $Z \sim Binomial(\kappa, (1 - e^{-\lambda\Delta}))$.

Thus, we have

$$\begin{aligned}
\mathbb{P}(Y \geq \kappa) &\geq \mathbb{P}(Z \geq \kappa) \\
&= \int_0^\infty \mathbb{P}(Z \geq \kappa | \Delta = t)\, f_\Delta(t) dt \\
&= \int_0^\infty (1 - e^{-\lambda t})^\kappa N\lambda e^{-N\lambda t} dt \\
&= \frac{N!\kappa!}{(N+\kappa)!} \\
&\geq (N+\kappa)^{-\kappa},
\end{aligned}$$

where $f_\Delta(t)$ is the probability density function of Δ.

Note that N is the number of users requesting a at server s, and is bounded by $N = O(\frac{\log n}{\log\log n})$, with high probability [19].

Now, we can finally prove (2). Let c' be an appropriate constant,

$$\begin{aligned}
\mathbb{P}\left(\cap_{k=1}^{\tau}(H_k = 1)\right) &= \mathbb{P}(H = 1)^\tau = (1 - \mathbb{P}(H = 0))^\tau \\
&= (1 - \mathbb{P}(Y \geq \kappa))^\tau \\
&\leq (1 - (N+\kappa)^{-\kappa})^\tau \\
&\leq (1 - (c'\frac{\log n}{\log\log n} + \kappa)^{-\kappa})^\tau,
\end{aligned}$$

[1]random variables $U \geq_d V$ if $\mathbb{P}(U > x) \geq \mathbb{P}(V > x)$ for all x.

which is $n^{-\Omega(1)}$ when $\tau = \Theta(\log^{\kappa+1} n)$. ■

By bounding the time for τ requests to arrive at user u, we have the following,

***Lemma** 2:* If user u (with candidate servers S_u) is not *decided* at time t, then the server is *overbooked* at time $t-\delta$ for $\delta = \frac{\tau+1}{\lambda} c_0$ where $c_0 > 1$ is a constant, with high probability.

Proof: Let random variable N_δ be the number of requests from u during time $(t-\delta, t)$, $N_\delta \sim Poisson(\lambda\delta)$. A bound on the tail probability of Poisson random variables is developed in [15] as

$$\mathbb{P}(X \le x) \le \frac{e^{-\lambda'}(e\lambda')^x}{x^x},$$

where $X \sim Poisson(\lambda')$ and $x < \lambda'$.

We can show there are at least $\tau+1$ requests during $(t-\delta, t)$ w.h.p. as the following,

$$\begin{aligned}\mathbb{P}(N_\delta < \tau+1) &\le e^{-\lambda\delta}\frac{(e\lambda\delta)^{\tau+1}}{(\tau+1)^{\tau+1}} = e^{-(\tau+1)c_0}(ec_0)^{(\tau+1)} \\ &= e^{-(\tau+1)(c_0-1)}c_0^{(\tau+1)} \\ &= n^{-\frac{(\tau+1)}{\log n}(c_0-1-\log c_0)} \\ &= n^{-\Theta(\log^\kappa n)},\end{aligned}$$

as $c_0 > 1$ and $\tau = \Theta(\log^{\kappa+1} n)$. Thus, w.h.p. no fewer than $\tau+1$ requests arrive at u. And because user is not decided at time t we know that with high probability, at least one of the previous τ requests results in a miss, which means that between the previous $(\tau+1)$-th request and the miss, κ different other requests arrived at the server. Thus server s is *overbooked* at the time the previous $(\tau+1)$-th request arrives, which with high probability is no earlier than $t-\delta$. ■

Based on Lemmas 1 and 2, we can then establish the following theorem about the performance of Algorithm *GoWithTheWinner*.

***Theorem** 3:* With probability at least $1 - \frac{1}{n^{\Omega(1)}}$, the minmax hit rate $H(t) = 100\%$ for all $t \ge T$, provided $\sigma \ge 2$ and $T = O(\frac{\kappa}{\log(\kappa+1)}(\log n)^{\kappa+1}\log\log n)$. That is, with high probability, algorithm *GoWithTheWinner* converges by time T to an optimal state where each user $u \in U$ has decided on a server $s \in S$ that serves it content with a 100% hit rate.

This is the main result for the performance analysis of the algorithm. Due to space limit, please refer to our technical report [13] for detailed proof of this theorem.

Are two or more random choices necessary for all users to receive a 100% hit rate? Analogous to the "power of two choices" in the balls-into-bins context [14], we show that two or more choices are required for good performance with the following theorem.

***Theorem** 4:* For any fixed constants $0 \le \alpha < 1$ and $\kappa \ge 1$, when algorithm GoWithTheWinner uses one random choice for each user ($\sigma = 1$), the minmax hit rate $H(t) = o(1)$, with high probability, i.e., $H(t)$ tends to zero as n tends to infinity, with high probability.

The reader is referred to technical report [13] for the proof.

D. When $n_u = n_s^\alpha, \alpha > 1$

Now we analyze the case that there are many more users than the number of servers. Assume $n_s = n, n_u = n^\alpha$ and $\kappa = \frac{n_u}{n_s} = n^{\alpha-1}$, we have the following result,

***Theorem** 5:* When $n_s = n, n_u = n^\alpha, \alpha > 1$, with probability at least $1 - \frac{1}{n^{\Omega(1)}}$, the maximum load (number of incoming servers) over all servers is $O(\sigma\frac{n_u}{n_s})$. Furthermore, if $\kappa = \frac{n_u}{n_s}$, all users have 100% hit rate.

Theorem 5 implies that when $n_u = n_s^\alpha$ all the servers have balanced load of $\sigma\frac{n_u}{n_s}$, and thus we don't need a server selection mechanism for load balancing other than just letting users randomly choose the server. In this case, it's not beneficial to let users start with more than one randomly selected servers, because with $\sigma = 1$ the load on all servers are balanced already. Thus, as long as we have feasible server capacity $\kappa = \omega(\frac{n_u}{n_s})$, all the users will have enough resources from the server and have 100% hit rate by randomly select one server.

The number of content items n_c here does not affect the result of load balancing. Actually, the result stays the same when $n_c \ge n_u$. When the number of content items is much smaller than number of users, $n_c << n_u$, the cache size can be made smaller because the number of distinct requests at each server becomes smaller.

III. Bitrate Maximization for Video Content

In video streaming, a key performance metric is the *bitrate* at which a user can download a video. If the server is unable to provide the required bitrate to the user, the video may frequently freeze resulting in an inferior viewing experience and reduced user engagement [11]. For simplicity, we model the server's bandwidth capacity that is often the critical bottleneck resource, while leaving other factors that could influence video performance such as the server-to-user connection and the server's cache[2] for future work.

A. Problem formulation

The bitrate required to play a stream without freezes is often the encoded bitrate of the stream. For simplicity, we assume that each user requires a bitrate of 1 unit for playing its video and each server has the capacity to serve an aggregation of κ units. We also assume each server evenly divides its available bitrate capacity among all users streaming videos from it. We assume each user can tell the exact bitrate that it receives from its chosen candidate servers and that this bitrate is used as the performance feedback (cf. Figure 1).

Unlike web content delivery, where users make random requests to the same website, we assume that users requesting video streaming maintain persistent connections with the server. We use a discrete time model in this case as compared to the continuous time model for web content delivery. We assume after each time unit that users examine the bit rate provided by each of the available servers and then make

[2]Unlike the web, cache hit rate is a less critical determinant of video performance. Videos are cached in chunks by the server. The next chunk is often prefetched from origin if it is not in cache, even while the current chunk is being played by the user, so as to hide the origin-to-server latency.

decisions according to the performance (measured by bit rate). The goal of each user is to find a server that can provide the required bitrate of 1 unit for viewing the video.

B. Algorithm MaxBitRate

After each user $u \in U$ has selected a video object $c_u \in C$ using the popularity distribution, Algorithm MaxBitRate described below is executed independently by each user $u \in U$, in discrete time steps.

1) Choose a random subset of candidate servers $S_u \subseteq S$ such that $|S_u| = \sigma$.
2) At each time step $t \geq 0$, do the following:
   a) Request the video content from all servers $s \in S_u$.
   b) For each server $s \in S_u$, compute $B(u, s, t) \triangleq$ bitrate provided by server s to user u in the current time step.
   c) If there exists a server $s \in S_u$ such that $B(u, s, t) = 1$, then *decide* on server s by setting $S_u \leftarrow \{s\}$.

Note that each user executes a simple strategy of trying σ randomly chosen servers initially. Then, using the bitrate received in the current time step as feedback, each user independently narrows it's choice of servers to a single server that provides the required unit bitrate. If multiple servers provide the required bitrate, the user selects one at random. Further, note that a user u downloading from a server s at time t knows immediately whether or not the server is overloaded, since server s is overloaded if user u received a bitrate of less than 1 unit from the server, i.e., $B(u, s, t) < 1$. This is a point of simplification in relation to the complex situation of hit rate maximization where any single cache hit is not indicative of a non-overloaded server and a historical average of hit rates over a large enough time window τ is required as a probabilistic indicator of server overload. Furthermore, this simplification yields both faster convergence to an optimal state in $T = O(\log\log n/\log(\kappa+1))$ steps and a much simpler proof of convergence.

C. Analysis of Algorithm MaxBitRate

As before, we rigorously analyze the case where $n_u = n_s = n$. Let the *minmax* bitrate $B(t)$ be the best bitrate obtained by the worst user at time t, i.e.,

$$B(t) \triangleq \min_{u \in U} \max_{s \in S} B(u, s, t).$$

***Theorem** 6:* When $\sigma \geq 2$, the minmax bitrate converges to $B(t) = 1$ unit, for all $t \geq T$, within time $T = O(\log\log n/\log(\kappa + 1))$, with high probability. When $\sigma = 1$ on the other hand, the minmax bitrate $B(t) = O(\kappa \log\log n/\log n)$, with high probability. In particular, when $\sigma = 1$ and the cache size κ is $o(\log n/\log\log n)$, including the case when κ is a fixed constant, $B(t)$ tends to zero as n tends to infinity, with high probability.

The proof can be found in [13].

IV. Empirical Evaluation

We empirically study our algorithm *GoWithTheWinner* through simulation. Requests from each user is modeled as a Poisson arrival sequence with unit rate. We use $n_u = 1000$ users. To simulate varying numbers of servers, users, and content items, we vary n_s and n_c such that $1 \leq n_u/n_c, n_u/n_s \leq 100$. We also simulate a range of values for the spread $1 \leq \sigma \leq 6$, and sliding window size $1 \leq \tau \leq 20$. Each server implements an LRU replacement policy of size $\kappa \geq 2$. We use the power law distribution for content popularity distribution, where the kth most popular object in C is picked with a probability

$$p_k \triangleq \frac{1}{k^\alpha \cdot \mathcal{H}(n_c, \alpha)}, \quad (4)$$

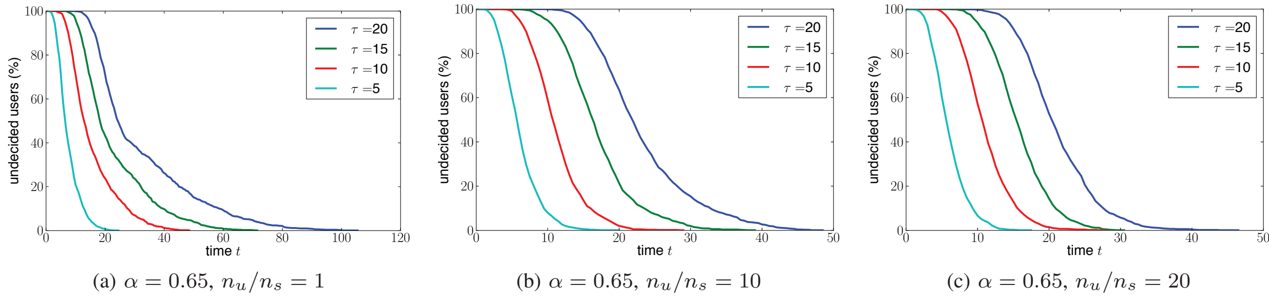
where $\alpha \geq 0$ is the exponent of the distribution and $\mathcal{H}(n_c, \alpha) = \sum_{k=1}^{n_c} 1/k^\alpha$. Note that power law distributions (aka Zipf distributions) are commonly used to model the popularity of online content such as web pages, and videos. This family of distributions is parametrized by a Zipf rank exponent α with $\alpha = 0$ representing the extreme case of an uniform distribution and larger values of α representing a greater skew in the popularity. It has been estimated that the popularity of web content can be modeled by a power law distribution with an α in the range from 0.65 to 0.85 [3], [9], [8]. In the simulations, the content items are requested by users using the power law distribution of (4) with $\alpha = 0.65$ to model realistic content popularity [3] [9]. However, we also vary α from 0 (uniform distribution) to 1.5 in some of our simulations.

The system *converges* when all users have decided on a single server from their set of candidate servers. There are two complementary metrics that relate to convergence. *Failure rate* is the probability that the system converged to a non-optimal state where there exists servers that are overbooked, resulting in some users incurring cache misses after convergence. The failure rate is calculated from multiple runs of the simulation. *Convergence time* is the time it takes for the system to converge provided that it converges to an optimal state.

A. Speed of convergence

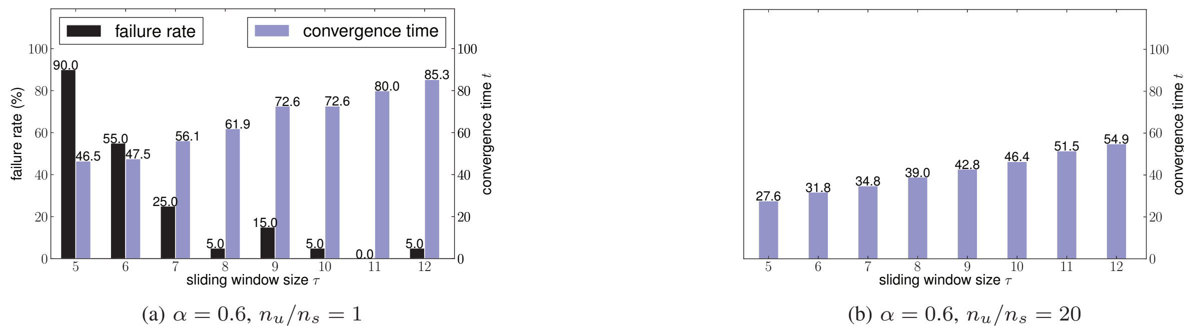
Figure 2 shows how the fraction of undecided users decreases over time until it reaches zero, resulting in convergence. Note that users do not decide in the first τ steps, since they must wait at least that long to accumulate a window of τ hits. However, once the first τ steps complete, the decrease in the number of undecided users is fast as users discover that at least one of their two randomly chosen candidate servers have less load. The rate of decrease in undecided users decreases towards the end, as the number of users who experience cache contention in *both* of their server choices require multiple iterations to resolve.

In this simulation, we keep the number of users $n_u = 1000$ but vary the number of servers n_s to achieve different values for n_u/n_s. Note that for a fair comparison, we keep the system-wide load the same. Load l is a measure of cache



(a) $\alpha = 0.65,\ n_u/n_s = 1$ (b) $\alpha = 0.65,\ n_u/n_s = 10$ (c) $\alpha = 0.65,\ n_u/n_s = 20$

Fig. 2: The figures show the percentage of undecided users for a typical power law distribution ($\alpha = 0.65$) with spread $\sigma = 2$ and $n_u = 1000$. Note that the undecided users decrease with time in all cases, but the convergence is faster when we use fewer but larger servers by setting n_u/n_s to be larger. Also, the smaller values of the look-ahead window τ result in faster convergence.



(a) $\alpha = 0.6,\ n_u/n_s = 1$ (b) $\alpha = 0.6,\ n_u/n_s = 20$

Fig. 3: Generally, as τ increases, convergence time increases but failure rate decreases. It is also true for larger servers (n_u/n_s = 20), only the failure has gone to zero for all investigated sliding window sizeτ.

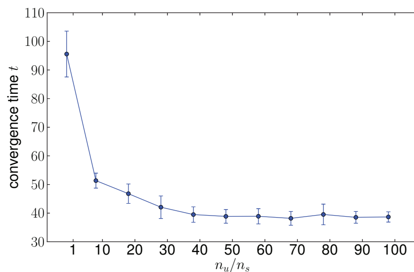


Fig. 4: As n_u/n_s increases fewer servers with larger capacity are used and convergence time decreases. The decrease is less pronounced beyond $n_u/n_s \geq 40$ under this setting ($\alpha = 0.65$, $\sigma = 2$, $\tau = 20$).

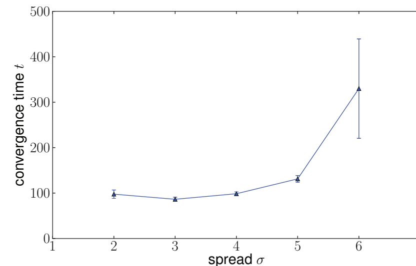


Fig. 5: There is a very small incremental benefit in using $\sigma = 3$ instead of 2, though higher values of $\sigma > 3$ only increased the convergence time. ($\alpha = 0.65, n_u/n_s = 1, \tau = 20, \kappa = 2$.)

contention in the network and is naturally defined as the ratio of the numbers of users in the system and total serving capacity that is available in the system. That is, $l \triangleq n_u/(\kappa \cdot n_s)$. For all three setting of Figure 2, we maintain a load $l = 0.5$. The figure shows that with fewer (but larger) servers (n_u/n_s is larger) the convergence time is faster, because having server capacity in a few larger servers provides a larger hit rate than having the same capacity in several smaller servers. Similar performance gains are also found in the context of web caching and parallel jobs scheduling [18]. Convergence times are plotted explicitly in Figure 4 for a greater range of user-to-server ratios. As n_u/n_s increases from 1 to 40, convergence time decreases. The decreases in convergence times are not significant beyond $n_u/n_s \geq 40$.

B. Impact of sliding window τ

The sliding window τ is the number of recent requests used by algorithm *GoWithTheWinner* to estimate the hit rate. As shown in Figure 3, there is a natural tradeoff between convergence time and failure rate. When τ increases, the users take longer to converge, as they require a 100% hit rate in a larger sliding window. However, waiting for a longer period also makes their decisions more robust. That is, a user is less

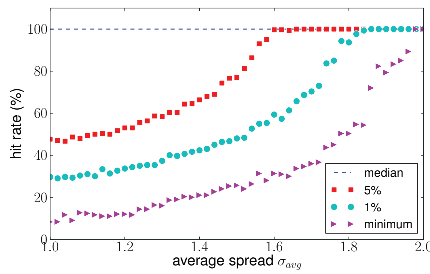


Fig. 6: Order statistics of the hit rate of the user population. ($\alpha = 0.65, n_u/n_s = 1, \tau = 10, \kappa = 2$.)

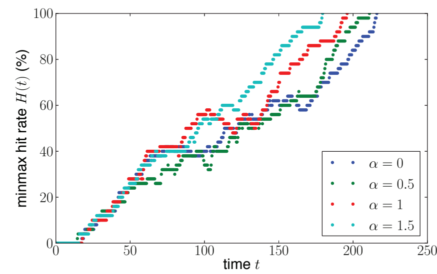


Fig. 7: Minmax hitrate versus time for different power law distributions.

likely to choose an overbooked server, since an overbooked server is less likely to provide a string of τ hits for large τ. In our simulations with many smaller caches ($n_u/n_s = 1$), when $\tau \leq 4$, users made quick choices based on a smaller sliding window. But, this resulted in the system converging to a non-optimal state 100% of the time. As τ further increases, failure rate decreases. The value of $\tau = 11$ is a suitable sweet spot as it results in the smallest convergence time for a zero failure rate. However, for fewer but larger servers ($n_s/n_u = 20$), all selections of window size τ (thus the small values like $\tau = 5$) yielded a 0% failure rate, while convergence time still increases as the window size gets larger.

C. Impact of spread σ

As shown in Theorems 3 and 4, a spread of $\sigma \geq 2$ is required for the system to converge to an optimal solution, while a spread of $\sigma = 1$ is insufficient. As predicted by our analysis, our simulations did not converge to an optimal state with $\sigma = 1$. Figure 5 shows the convergence time as a function of spread, for $\sigma \geq 2$.

As σ increases, there are two opposing factors that impact convergence time. The first factor is that as σ increases, each user has more choices and a user is more likely to find a suitable server with less load. On the other hand, an increase in σ also increases the total number of initial requests in the system that equals σn_u. Thus, the initiate server load increases in σ. These opposing forces result in a very small incremental benefit when using $\sigma = 3$ instead of 2, though the higher values of $\sigma > 3$ showed no benefit as convergence time increases with σ increases.

We established the "power of two random choices" phenomenon where two or more random server choices yield superior results to having just one. It is intriguing to ask *what percentage of users need two choices* to reap the benefits of multiple choices? Consider a mix of users, some with two random choices and others with just one. Let σ_{avg}, $1 \leq \sigma_{avg} \leq 2$, denote the average value of the spread among the users.

In Figure 6, we show different order statistics of the hit rate as a function of σ_{avg}. Specifically, we plot the minimum value, 1st-percentile, 5th- percentile and the median (50th-percentile) of user hit rates after simulating the system for 200 time units. As our theory predicts, when $\sigma_{avg} = 2$, the minimum and all the order statistics converge to 100%, as all users converge to a 100% hit rate. Further, if we are interested in only the median user, any value of the spread is sufficient to guarantee that 50% of the users obtain a 100% hit rate. Perhaps the most interesting phenomena is that if $\sigma_{avg} = 1.7$, i.e., 70% of the users have two choices and the rest have one choice, the 5th-percentile converges to 100%, i.e., all but 5% of the users experience a 100% hit rate. For a higher value of $\sigma_{avg} = 1.9$, the 1st-percentile converges to 100%, i.e., all but the 1% of the users experience a 100% hit rate. This result shows that our algorithm still provides benefits even if only *some* users have multiple random choices of servers available to them.

D. Impact of demand distribution

We now study how hit rate changes with the exponent α in the power law distribution of Equation 4. Note that the distribution is uniform when $\alpha = 0$ and is the harmonic distribution when $\alpha = 1$. As α increases, since the tails fall as a power of α, the distribution becomes more skewed towards content items with a smaller rank. In Figure 7, we plot the minmax hitrate over time for different α, where we see that a larger α leads to faster convergence. The reason is that as the popularity distribution gets more skewed, a larger fraction of users will request the same popular content items, leading to higher hit rate and faster convergence. Thus, the uniform popularity distribution ($\alpha = 0$) is the worst case and the algorithm converges faster for the distributions that tend to occur more commonly in practice. Providing theoretical support for this empirical result by analyzing the convergence time to show faster convergence for larger α is a topic for future work.

V. Related work

Server selection algorithms have a rich history of both research and actual implementations over the past two decades. Several server selection algorithms have been proposed and empirically evaluated, including client-side algorithms that use historical performance feedback using probes [7], [5]. Server selection has also been studied in a variety of contexts, such as the web [5], [20], video streaming [21], and cloud services [22]. Our work is distinguished from the prior literature in that we theoretically model the "Go-With-The-Winner" paradigm

that is common to many proposed and implemented client-side server selection algorithms. Our work is the first formal study of the efficacy and convergence of such algorithms.

In terms of analytical techniques, our work is closely related to prior work on balls-into-bins games where the witness tree technique was first utilized [14]. Witness trees were subsequently used to analyze load balancing algorithms, and circuit-switching algorithms [4]. However, our setting involves additional complexity requiring novel analysis due to the fact that users can share a single cached copy of an object and the hitrate feedback is only a probabilistic indicator of server overload. Also, our work shows that the "power of two random choices" phenomenon applies in the context of content delivery, a phenomenon known to hold in other contexts such as balls-into-bins, load balancing [23], relay allocation for services like Skype [16], and circuit switching in interconnection networks [14].

VI. Conclusion

Our work constitutes the first formal study of the simple "Go-With-The-Winner" paradigm in the context of web and video content delivery. For web (resp., video) delivery, we proposed a simple algorithm where each user randomly chooses two or more candidate servers and selects the server that provided the best hit rate (resp., bitrate). We proved that the algorithm converges quickly to an optimal state where all users receive the best hit rate (resp., bitrate) and no server is overloaded, with high probability. While we make some assumptions to simplify the theoretical analysis, our simulations evaluate a broader setting that incorporates a range of values for τ and σ, varying content popularity distributions, differing load conditions, and situations where only some users have multiple server choices. Taken together, our work establishes that the simple "Go-With-The-Winner" paradigm can provide algorithms that converge quickly to an optimal solution, given a sufficient number of random choices and a sufficiently (but not perfectly) accurate performance feedback.

VII. Acknowledgements

This work was supported in part by NSF grant CNS-1413998. It was partially sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *INFOCOM, 2012 Proceedings IEEE*, pages 1620–1628. IEEE, 2012.

[2] Akamai. Akamai download manager. 2013. http://www.akamai.com/html/solutions/downloadmanager_overview.html.

[3] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, 1999.

[4] R. Cole, B. M. Maggs, M. Mitzenmacher, A. W. Richa, K. Schröder, R. K. Sitaraman, B. Vöcking, et al. Randomized protocols for low-congestion circuit routing in multistage interconnection networks. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 378–388. ACM, 1998.

[5] M. E. Crovella and R. L. Carter. Dynamic server selection in the internet. Technical report, Boston University Computer Science Department, 1995.

[6] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *Internet Computing, IEEE*, 6(5):50–58, 2002.

[7] S. G. Dykes, K. A. Robbins, and C. L. Jeffery. An empirical evaluation of client-side server selection algorithms. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1361–1370. IEEE, 2000.

[8] C. Fricker, P. Robert, and J. Roberts. A versatile and accurate approximation for lru cache performance. In *Proceedings of the 24th International Teletraffic Congress*, ITC '12.

[9] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization: A view from the edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07. ACM, 2007.

[10] P. Key, L. Massoulie, and P. Towsley. Path selection and multipath congestion control. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007.

[11] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 211–224. ACM, 2012.

[12] C. Liu and P. Albitz. *DNS and Bind*. O'Reilly Media, Inc., 2009.

[13] C. Liu, R. K. Sitaraman, and D. Towsley. Go-with-the-winner: Client-side server selection for content delivery. *CoRR*, abs/1401.0209, 2014.

[14] M. Mitzenmacher, A. W. Richa, and R. Sitaraman. The power of two random choices: A survey of techniques and results. *COMBINATORIAL OPTIMIZATION-DORDRECHT-*, 9(1):255–304, 2001.

[15] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[16] H. X. Nguyen, D. R. Figueiredo, M. Grossglauser, and P. Thiran. Balanced relay allocation on heterogeneous unstructured overlays. In *INFOCOM*, 2008.

[17] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[18] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM.

[19] M. Raab and A. Steger. balls into binsa simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*, pages 159–170. Springer, 1998.

[20] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithms for replicated web servers. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):44–50, 1998.

[21] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao. Dissecting video server selection strategies in the youtube cdn. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 248–257. IEEE, 2011.

[22] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. Donar: decentralized server selection for cloud services. In *ACM SIGCOMM Computer Communication Review*, volume 40. ACM, 2010.

[23] L. Ying, R. Srikant, and X. Kang. The power of slightly more than one sample in randomized load balancing. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015.