# Approximate Server Selection Algorithms in Content Distribution Networks

Spiridon Bakiras
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
sbakiras@cs.ust.hk

*Abstract*— **Server selection is an important function in any replication-based infrastructure, aiming at redirecting client requests to the "best" server according to some predefined metrics. Previous research work has mainly focused on client-side redirection schemes, where the client is responsible for the server selection process. Furthermore, previous work has shown that client probing techniques perform significantly better in discovering the "best" server, compared to hop- or RTT-based schemes. Client probing, however, is not very scalable, since the number of clients and servers in the network will be very large. In this paper, we propose a novel technique to transform the server selection problem into a problem of optimal routing, which enables us to shift the redirection process from the client to the server-side. In particular, we consider the environment of a Content Distribution Network (CDN), and propose a flexible framework that can be used to optimize the server selection process, according to various metrics and/or policies. Using trace-driven simulations, we show that the proposed method can improve significantly the response time of HTTP requests while keeping the control overhead at a very low level.**

## I. Introduction

The explosive growth of the World Wide Web and the increasing availability of fast Internet access to the end-user, have turned centralized web servers into a performance bottleneck. Popular web sites (e.g., news sites) receive tens of millions of requests per day, which may easily overload a state-of-the-art web server and increase significantly the delay perceived by end-users.

Replication is a popular technique for reducing the latency of HTTP requests, and is realized by moving the web content as close to the end-user as possible. Content distribution networks (CDNs), for example, accomplish that by replicating popular web sites across a number of geographically distributed servers. The key objectives of a CDN are to increase the availability of the hosted sites and, most importantly, to minimize the response time of HTTP requests. One of the most important functions in any replication-based infrastructure is the server selection process, i.e., the process of identifying the "best" server for each client, according to some predefined metrics. Previous research work has mainly focused on client-side redirection schemes, where the client is responsible for the server selection process. Furthermore, previous work has shown that client probing techniques perform significantly better in discovering the "best" server, compared to hop- or RTT-based schemes. Client probing, however, is not very scalable, since the number of clients and servers in the network will be very large.

In this paper, we propose a novel technique to transform the server selection problem into a problem of optimal routing, which enables us to shift the redirection process from the client to the server-side. In particular, we consider the environment of a CDN system, and propose a flexible framework that can be used to optimize the server selection process according to various metrics and/or policies. Using trace-driven simulations, we show that the proposed method can improve significantly the response time of HTTP requests while keeping the control overhead at a very low level.

The remainder of the paper is organized as follows. In Section II we give a brief overview of previous research work on server selection algorithms. The proposed transformation technique is introduced in Section III, while the simulation results are illustrated in Section IV. Finally, Section V concludes our work.

## II. Previous Work

Server selection algorithms may be generally classified into two categories, depending on which metrics they take into consideration for selecting the "best" server. The first category consists of those algorithms that use network distance as the selection criteria, and typical examples include [8], [5], [10], [11]. Guyton and Schwartz [8] use routing table polling and network probing to calculate the distance between a client-server pair. Similarly, the work by Jamin et al. [10] utilizes the IDMaps [7] infrastructure to derive estimates of the client-server distance. In a more recent work, Ratnasamy et al. [11] introduce a binning scheme to cluster nodes into bins, such that the nodes inside any bin are relatively close to each other in terms of network latency. Finally, a slightly different approach is considered by Carter and Crovella [5], which uses several tools to measure the latency and available bandwidth from the server to the client, in order to make the selection process more dynamic.

The other class of server selection algorithms concentrates on the complete path from the server to the client that includes both the server load and the network delay. Sayal et al. [12] show that the correlation between either the number of hops or the measured RTT (through the *ping* utility) and the response time is quite low. Instead, the best policy is shown to be past

0-7803-8938-7/05/$20.00 (C) 2005 IEEE

latency, an estimate of which is kept by sending periodic HTTP HEAD messages to the server. Similar results are reported by Dykes et al. [6], where the dynamic probe approach is again shown to be superior to other metrics, such as bandwidth or latency. A different two-step selection metric is introduced by Hanna et al. [9], where a small subset of five well-performing servers is isolated from the server population, and testing is restricted to that subset for a period of ten days. Then, each request is redirected randomly to any one of these five servers, thus achieving some degree of load balancing.

Finally, the studies in [1] and [13] are more closely related to our work. Andrews et al. [1] introduce a system called Webmapper, which clusters the clients according to their IP addresses, and assigns each cluster to an optimal server. The clustering is performed by monitoring the TCP connections between the clients and the servers. When the best server for a cluster is discovered (by solving a min-cost flow problem), this information is propagated to the clients through the DNS infrastructure. The work by Zegura et al. [13] proposes an application-layer anycasting architecture, where anycast resolvers are distributed across the network and maintain metric information needed by the clients. The authors consider two different techniques, namely server push and client probing, that offer different levels of accuracy and overhead. However, a hybrid approach that combines the two is shown to be the best policy, since it offers a good trade-off between metric accuracy and scalability.

## III. Server Selection Algorithms

### A. System Model

We assume that the CDN infrastructure consists of $N$ geographically distributed servers, where each server $i$ has a processing capacity $C_i$ (in bps). We also assume that there are $M$ different objects (e.g., web sites) that are being replicated at various servers inside the CDN provider's network.

Co-located with each server is a redirection agent, which is responsible for redirecting the client requests to the appropriate CDN server. Whenever a client issues an HTTP request for one of the $M$ hosted objects, the DNS resolver at the client side will reply with the IP address of the nearest, in terms of network distance, server. Based on the redirection matrix, the redirection agent at that server will route the packet towards the appropriate CDN server (if necessary). The HTTP reply will be forwarded directly to the client.

The client population behind a server $j$ will generate a certain amount of requests for each of the $M$ objects. Let us use $r_{ij}$ to denote the traffic load (in bps) from object $i$ towards the clients of server $j$. Then, the traffic load generated from object $i$ will be equal to $r_i = \sum_{j=1}^{N} r_{ij}$, while the overall load from all the hosted objects will be equal to $r = \sum_{i=1}^{M} r_i$. In this work we try to answer the following question: given a replica placement matrix (i.e., the detailed location of copies) and the corresponding traffic load matrix (i.e., the values of $r_{ij}$), where should each client request be redirected in order to minimize the average response time for the whole system?

### B. Problem Transformation

Consider the network topology shown in Fig. 1. It consists of four columns of nodes, where the leftmost column represents the $M$ hosted objects, and each of the other three columns represents the $N$ servers of the CDN architecture. Therefore, each server $i$ $(1 \leq i \leq N)$ has three instances in the topology and, in particular, it is represented by nodes $(M + i)$, $(M + N + i)$ and $(M + 2N + i)$. Traffic will enter the topology through the $M$ objects, and exit at the servers of the rightmost column. Consequently, the set of origin-destination (OD) pairs will consist of $MN$ elements. For each OD pair there will be at most $N$ distinct paths, which represent the $N$ different servers that may satisfy a client request. For example, a positive amount of flow on the path $i \rightarrow (M+j) \rightarrow (M+N+j) \rightarrow (M+2N+k)$ indicates that some requests for object $i$ generated by a client in the area of server $k$, will be redirected to server $j$.

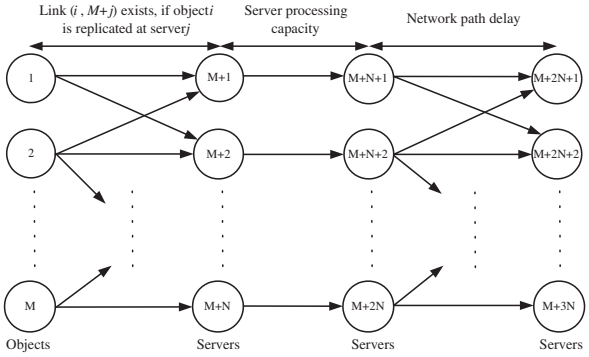



Fig. 1. Transformation of the server selection problem into a problem of optimal routing.

Our goal is to minimize the delay inside this artificial network, using a well-known optimal routing technique. In particular, we choose the gradient projection method, using the following cost function for each link $(i, j)$ [3]

$$D_{ij} = \frac{F_{ij}}{C_{ij} - F_{ij}} + d_{ij} F_{ij} \tag{1}$$

where $C_{ij}$ is the link capacity, $d_{ij}$ is the propagation and processing delay, and $F_{ij}$ is the amount of traffic on that link.

Let us now examine closer the various links of the topology in Fig. 1, and identify their role in the response time of user requests. First, the links on the left hand side of the topology correspond to the different servers where a particular object is replicated. In other words, a link $(i, M+j)$ will be present in the topology only if object $i$ is replicated at server $j$. However, the value of $F_{ij}$ should not affect the latency of client requests, since these links are not part of the data path. This fact may be realized by setting $d_{ij} = 0$ and $C_{ij} \gg r$ in Equation (1).

The bottleneck links in the middle of the topology correspond to the delay encountered at the CDN servers. It consists of two parts, namely, the processing delay (e.g., due to disk access time), and the delay due to the server load. In general, the second part (i.e., the term $F_{ij}/(C_{ij} - F_{ij})$ in Equation (1))

will dominate the average delay, especially when the server is highly loaded.

Finally, the links on the right hand side of the network topology model the average delay experienced by the HTTP reply message inside the core network. However, the HTTP reply messages destined towards different clients within the area of a single server, will follow different routes inside the core network. As a result, it is impossible to estimate this delay on a per-client basis. Instead, we follow a simple approach in which the redirection agents periodically exchange *ping* messages (e.g., every few minutes), thus obtaining a rough estimate of the network delay (both propagation and queueing) between any pair of servers. Since we assume that the clients will always contact the closest (in terms of network distance) server, this delay will also approximate the actual delay between any client-server pair. Going back to Equation (1), we may set $C_{ij} \gg r$, and $d_{ij}$ to be equal to the delay measured by the *ping* messages.

Our problem formulation is very flexible, and can be easily modified to solve different versions of the server selection problem. For instance, if we decide that load balancing is more important than average delay, we may set $d_{ij} = 0$ in all the links of the topology. Furthermore, the CDN provider may explicitly favor some servers to satisfy the requests for a specific object (i.e., policy-based redirection). This approach may be realized by setting $d_{ij} > 0$ in any link $(i, M + j)$ on the left hand side of the topology, if we want to discourage requests for object $i$ being redirected to server $j$.

Finally, the computational complexity of the gradient projection method (and thus of our approach) is $O(LMN^2)$, where $L$ is the number of iterations. However, the projection method converges very fast to a good solution, meaning that the value of $L$ would normally be very small. For instance, in all our simulation experiments the gradient projection method converged at most within a couple of iterations.

### C. Implementation Details

The solution obtained from the optimal routing formulation is a global redirection matrix, containing the selection probabilities for each object-server pair. In order to adapt to the changing access patterns and avoid the overloading of the servers, this matrix should be updated, if necessary, based on the most recent traffic load matrix (i.e., the values of $r_{ij}$). The calculation of the redirection matrix will be performed in a completely distributed manner at the redirection agents of the CDN infrastructure. Specifically, each agent will propagate its own traffic load vector to all the other agents, and the updated version of the redirection matrix will be calculated locally at every agent. Notice, that there is no reason to propagate any information regarding the network delay, since each redirection agent keeps an estimate of the delay towards every other agent.

The most important issue regarding the operation of the redirection mechanism, is when to update the redirection matrix. The obvious solution would be to perform the updates at regular time intervals (e.g., every five minutes). This method, however, has two disadvantages: (i) it will incur unnecessary overhead at periods when the traffic load matrix remains fairly stable, and (ii) it will perform poorly when the access patterns are changing very fast (unless, of course, we set the update interval to a very small value). Instead, we propose a technique where the updates are triggered by the CDN servers as soon as they discover a potential change in the access patterns.

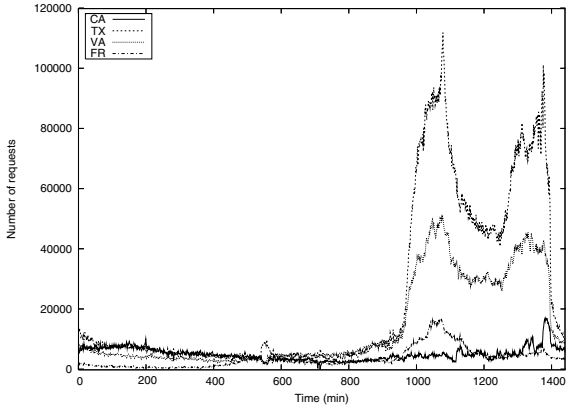


Fig. 2. Workload for day 62 of the WorldCup98 logs.

A straightforward indication of a varying traffic load matrix, would be a significant change in the utilization at one or more servers. This change would also affect the overall performance of the system, as it is evident in Equation (1). In order to maintain a low rate of updates, we set certain thresholds that need to be violated before a server can trigger an update. Specifically, each server will measure periodically (every one minute) its current utilization $u_{new}$, and compare it to a previous value $u_{old}$. If the two values differ by at least $\delta$, the server will trigger an update and broadcast its local traffic load matrix. Notice, that a similar technique may be applied in the case of the network delay part ($d_{ij}$) of the optimal routing formulation. Whenever a server discovers (through the *ping* messages) a significant change in the network delay towards another server, it may also initiate an update procedure.

Clearly, the control overhead imposed by our server selection algorithm is very low, in terms of both network and server load. Unlike client-side redirection schemes where client probing is essential, our approach is based on a global optimization of the server selection probabilities, which is transparent to the end-user. Consequently, the only overhead of our algorithm is due to the exchange of the ping messages and the traffic load vectors among the redirection agents, both of which are performed in the order of minutes.

## IV. Simulation Experiments

### A. Simulation Setup

*Dataset*: We used one day from the WordCup98 server logs [2] to feed our simulations. Specifically, we selected day 62, which was one of the busiest days with a total of over 68 million requests. The requests were collected from four different servers across the United States and Europe, located at California (CA), Texas (TX), Virginia (VA) and France

(FR). Of particular interest to our work is the fact that the server workload varies significantly both across the different servers and over time, as shown in Fig. 2.

*Network topology*: Following the system model described in Section III-A, we assume that the clients issuing the requests are located within a small network distance (around 40 ms round-trip) from the corresponding server. In addition, the inter-server delays were set in accordance to the geographic location of the four servers. We also consider the case of homogeneous servers, i.e., all the servers have identical processing capacity $C$, and processing delay $d = 20$ ms. Consequently, the CDN architecture consists of $N = 4$ servers, and is required to provide a hosting service to $M = 1$ object. Finally, we assume that all the requested documents are of fixed size (equal to $12KB$), and that the requests at the servers are processed in a FIFO order.

In order to test the performance of our algorithm in a larger topology, we also used the GT-ITM topology generator [4] to generate a random transit-stub graph. We set the number of stub domains to be equal to 50, and placed one CDN server inside each stub domain (i.e., $N = 50$). In addition, we set the propagation, queueing and processing delay inside the network to be equal to 20 ms/hop. Similarly, we split the original dataset into 50 synthetic datasets, by assigning each request to a particular server according to some probability. We run two sets of experiments, by drawing these probabilities from two different distributions. The first one was a Gaussian distribution with mean $\mu = 1/N$ and standard deviation $\sigma = 1/4N$, while the second was an exponential distribution with mean $\mu = 1/N$.

### B. Simulation Results

In this section we compare the performance, in terms of user-perceived latency, of the following four server selection algorithms

- *Approximate*: This is our proposed server-side selection algorithm, based on the gradient projection method.
- *Proximity*: Each request is served at the nearest server (i.e., the server where the request was initially forwarded to).
- *Random*: Each request is redirected to a randomly selected server.
- *Delay*: This is a non-realistic case, where we assume that perfect knowledge (regarding the individual server loads and the network path delay) is available, and each request is redirected to the server that would result in the minimum delay (at that time instant).

Fig. 3 shows the cumulative distribution function (CDF) of the response time for the four server selection strategies. The server capacity was set to $C = 80$ Mbps, which corresponds to approximately 800 HTTP requests/sec. The proximity-based approach has the worst performance overall, mainly due to the fact that server processing capacity at TX is not enough to accommodate the high arrival rate during the two peaks shown in Fig. 2. Furthermore, random redirection performs much worse than the approximate method, and its corresponding

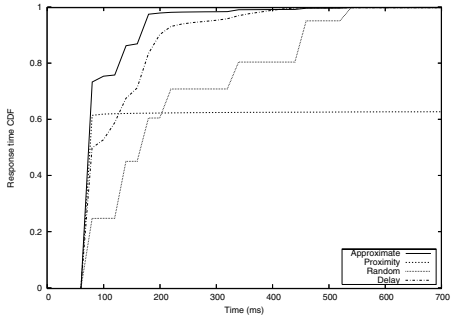
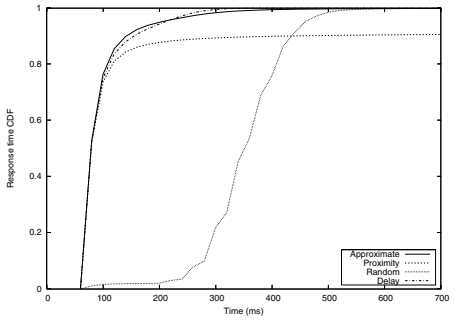
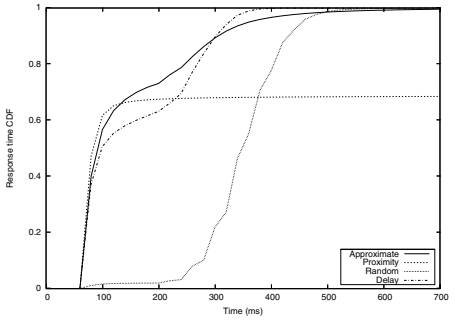



Fig. 3. Response time CDF for different server selection mechanisms (real dataset, $C = 80$ Mbps).




(a) Gaussian.




(b) Exponential.

Fig. 4. Response time CDF for different server selection mechanisms (synthetic dataset, $C = 6.5$ Mbps).

average response time is more than 2 times larger. This is due to the fact that network distance is not taken into account during the server selection process. Finally, the "Delay" curve in Fig. 3 illustrates a potential problem of client-side redirection. Specifically, once a new best server is discovered, all the subsequent client requests are forwarded to that server, causing it to overload for a small period of time (oscillation effect). This trend was also observed in [13], where the authors used the concept of *equivalent servers* (i.e., a set of servers with similar performance) in order to avoid it. The optimal routing formulation, though, overcomes this limitation by assigning different selection probabilities to each server.

Similar results hold for the synthetic datasets, and the corresponding graphs are shown in Fig. 4. The only difference is in the performance of the minimum delay approach, which

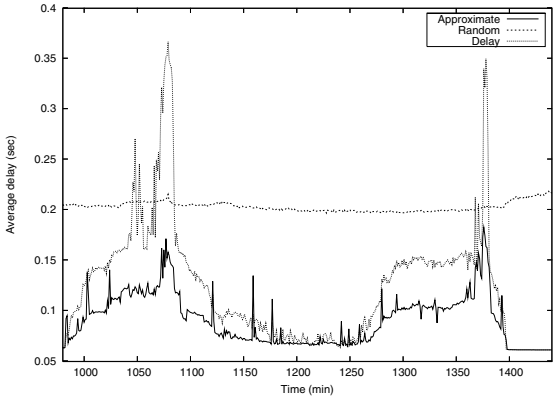



Fig. 5. Average response time for different server selection mechanisms (real dataset, $C = 80$ Mbps).

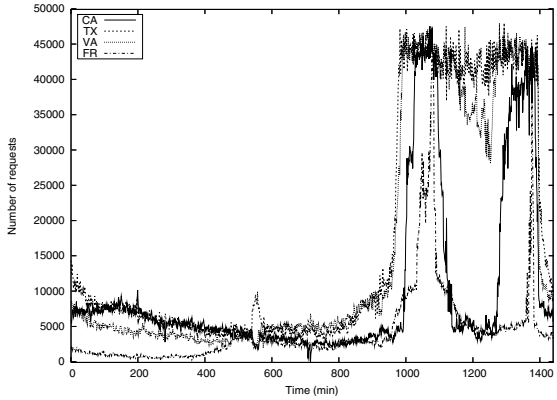



Fig. 6. Server load as a function of time (real dataset, $C = 80$ Mbps).

is improved considerably. This is due to the lower arrival rate of requests at each server that minimizes the oscillation effect. Furthermore, the synthetic dataset produced by the Gaussian distribution has the best performance overall, since the variance in the arrival rate among the CDN servers is very small. Also notice that we have scaled the server capacity $C$, in order to maintain a cumulative capacity equivalent to the real dataset case.

A more detailed picture of the dynamics of each technique is given in Fig. 5. In particular, this figure shows the average delay experienced by the end-users during the last 8 hours of the trace (i.e., during the two peaks). The random selection strategy has a very stable performance, and the average response time remains almost constant at around 200 ms. The approximate method experiences many small peaks that are caused by the changes in the access patterns. However, these peaks are eliminated very fast, since they trigger an update of the redirection matrix. Finally, the minimum delay approach performs considerably worse than the approximate method, due to the instability problem discussed earlier. The results for the synthetic datasets are consistent with the real dataset, and are omitted due to lack of space.

Finally, Fig. 6 illustrates the ability of the gradient projection algorithm to make the "correct" redirection decisions (the reader should also refer to Fig. 2 for comparison). When the processing capacity at the servers is sufficient (i.e., just before the two peaks), all the requests are served locally at the corresponding servers. When the offered load is increased (mainly at the servers in TX and VA), some requests are redirected to nearby servers, in order to accommodate the overall arrival rate. In addition, the fraction of requests to be redirected depends both upon the processing capacity, and the network distance among the different servers. Specifically, the majority of the "overflow" requests are redirected to the server in CA, and only a small fraction of them is redirected to FR, due to the large network delay.

## V. Conclusions

In this paper, we proposed a novel technique to transform the server selection process, in the context of a generic CDN system, into a problem of optimal routing. Specifically, we introduced a flexible framework based on server-side redirection, that can be used to optimize the server selection process according to various metrics and/or policies. Our simulation results indicate that the proposed method can improve significantly the response time of HTTP requests, and it outperforms all strategies that try to minimize other parameters, such as network distance or delay. Furthermore, the control overhead imposed by this approach is very low, both in terms of network and server load.

## References

[1] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane, "Clustering and server selection using passive monitoring," in *Proc. IEEE INFOCOM*, June 2002, pp. 1717–1725.

[2] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," HP Laboratories, Technical Report HPL-1999-35R1, 1999.

[3] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1992.

[4] K. Calvert and E. Zegura, "GT Internetwork Topology Models (GT-ITM)," Available at: http://www.cc.gatech.edu/projects/gtitm/.

[5] R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proc. IEEE INFOCOM*, April 1997, pp. 1014–1021.

[6] S. G. Dykes, K. A. Robbins, and C. L. Jeffery, "Empirical evaluation of client-side server selection algorithms," in *Proc. IEEE INFOCOM*, March 2000, pp. 1361–1370.

[7] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewicz, and Y. Jin, "An architecture for a global internet host distance estimation service," in *Proc. IEEE INFOCOM*, March 1999, pp. 210–217.

[8] A. Guyton and M. Schwartz, "Locating nearby copies of replicated internet servers," in *Proc. ACM SIGCOMM*, August 1995, pp. 288–298.

[9] K. M. Hanna, N. Natarajan, and B. N. Levine, "Evaluation of a novel two-step server selection metric," in *Proc. IEEE International Conference on Network Protocols (ICNP)*, November 2001, pp. 290–300.

[10] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of internet instrumentation," in *Proc. IEEE INFOCOM*, March 2000, pp. 295–304.

[11] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. IEEE INFOCOM*, June 2002, pp. 1190–1199.

[12] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection algorithms for replicated web servers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 44–50, 1998.

[13] E. W. Zegura, M. H. Ammar, Z. Fei, and S. Bhattacharjee, "Application-layer anycasting: a server selection architecture and use in a replicated web service," *IEEE/ACM Transactions on Networking*, vol. 8, no. 4, pp. 455–466, August 2000.