Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 29 (2006) 1026–1038
computer communications
www.elsevier.com/locate/comcom

Dynamic server selection using fuzzy inference in content distribution networks

Lin Cai, Jun Ye, Jianping Pan, Xuemin (Sherman) Shen*, Jon W. Mark

Department of Electrical and Computer Engineering, Centre for Wireless Communications, University of Waterloo, 200 University Ave. W., Waterloo, Ont., Canada N2L 3G1

Received 13 September 2004; revised 17 May 2005; accepted 3 June 2005
Available online 29 June 2005

Abstract

To accommodate the exponential growth of Web traffic, Content Distribution Networks (CDN) have been designed and deployed to distribute content to different cache servers, and to transparently and dynamically redirect user requests to the cache servers according to the latest network and server status. Server selection therefore is vital and crucial to both the functionality and performance of any CDN systems. An appropriate server should be selected by taking estimated user location, measured round-trip time, and advertised server load into account. However, it is unlikely to obtain accurate and timely inputs of these parameters in practice, so that the effectiveness and efficiency of CDN cannot be fully achieved by traditional means. In this paper, a novel CDN server selection scheme using fuzzy inference is proposed. The scheme selects appropriate servers based on partial round-trip time measurements and historical server load information, and it can be implemented generically wherever the decision is made. It is shown that the fuzzy inference-based scheme is inherently capable of handling multiple decision inputs efficiently, tolerable to measurement noise and errors, and able to deal with network dynamics. Simulation results demonstrate that, compared with other server selection schemes, the proposed scheme can achieve higher resource utilization, provide better user-perceived Quality of Service (QoS), and efficiently deal with network dynamics.

Keywords: Computer network performance; Fuzzy logic; Content distribution; Server selection

1. Introduction

Content Distribution Networks (CDN) and Web caching have attracted intensive attention in recent years [1] because of their capabilities of alleviating the heavy burden sustained by many popular client–server Internet applications. By introducing CDN cache servers or Web proxy servers between an origin server and its users (or clients), CDN and Web caching can avoid server overload, reduce network traffic, and improve user-perceived online experience considerably. This gives network and service providers a great incentive to quickly adopt and deploy CDN and Web caching in a competitive market.

Compared with Web caching where the proxy server assigned for a *user* is almost static, CDN has the flexibility to choose a *good* cache server transparently for an individual *user request* according to the latest network and server status. When a cache server fails or under-performs, or its network path becomes unavailable or congested, CDN has the capability of dynamically redirecting or offloading requests to other cache servers. It is worth mentioning that CDN and Web caching can work together harmonically. For example, requests from neighboring users can first be aggregated at a nearby proxy server; if a proxy cache miss occurs, the regenerated request is handled by a CDN cache server. *Server selection* therefore is vital and critical to both the functionality and performance of any CDN-related systems, and selection schemes should be designed carefully to balance the trade-off between the introduced overhead and possible performance gains. This task becomes more complicated when such a scheme needs to be transparent to end users and adaptive to network and server dynamics. The transparency is achieved by out-of-band Domain Name System (DNS) resolution, in-band HTTP redirection, or other means. The selection adaptivity should consider the following factors: (i) user–server proximity in a CDN;

* Corresponding author. Tel.: +1 519 888 4567x2691; fax: +1 519 746 3077.
E-mail address: xshen@bbcr.uwaterloo.ca (X.(S.) Shen).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.06.001

(ii) server load; or (iii) network connectivity between user and server [2]. Intuitively, CDN chooses a nearby, lightly loaded, and reachable cache server for a user request. In addition, the server selection should meet the following criteria: (i) appropriate from user's viewpoint, since the selection is made by CDN remotely; (ii) lightweight in terms of extra overhead introduced at both CDN and user sides; (iii) easy to develop, evaluate, and deploy. However, the existing Internet infrastructure offers little help for CDN server selection. Both users and servers are only identified by their IP addresses, which have very weak correlation with their actual location in geographical or network space. Server load is difficult to measure and update when there is a large number of servers distributed across the network. The packet-switching nature and traffic dynamics over the Internet make an accurate measurement of network connectivity between user and server almost infeasible. Moreover, it is still unclear how to synthesize server load and network connectivity, which are usually measured separately with different metrics in practice.

There are several CDN server selection schemes that are based on a deterministic decision process. For instance, a server is chosen for a user request since the server is the closest one (e.g. hosted in a nearby city or sharing the longest IP address prefix with the user), or is the least loaded one (e.g. with the highest capacity or lowest number of active requests), or has the best network connectivity (e.g. with the smallest router hop count or round-trip time, *rtt*, from the user). However, many practical issues prevent the effectiveness and efficiency of these deterministic server selection schemes. First, it is very difficult, if not impossible, to obtain an accurate measurement of these decision inputs (e.g. server load, *rtt*, and so on) in a timely manner. Second, even if a relatively accurate measurement is available, it may not offer enough granularity for comparison (e.g. client–server router hop count tends to cluster in a narrow range). In fact, the measurement on server load and network connectivity contains considerable noise or even errors. Although some sophisticated deterministic schemes can consider multiple metrics by assigning a weight to each of them beforehand [3], such a static approach cannot deal with network dynamics effectively and efficiently. Overall, the challenge in this context is how to make an appropriate decision based on *multiple*, *inaccurate*, and *inconsistent* inputs. In addition, the decision process may become location-dependent, since each decision maker only has a local view of the global network. This fact may further complicate the implementation of a deterministic scheme.

In this paper, we propose a fuzzy inference-based scheme for CDN server selection. The proposed scheme has the following features: (i) inherently capable of handling multiple decision inputs efficiently, no matter whether these inputs are implicitly or explicitly related; (ii) intrinsically tolerable to noise and errors in these measured decision inputs—it can also perform reasonably well even when some of these inputs are temporarily unavailable; (iii) highly adaptive to network and server dynamics when compared with other schemes using static weights for different metrics; (iv) generically implementable with independent trainings for individual decision makers to build their inference rule-bases. Moreover, a well-trained inference engine is less vulnerable to system oscillation (also know as herd effect in distributed systems, e.g. when a server is known lightly loaded, it quickly becomes overwhelmed by all redirected requests due to a deterministic server selection scheme).

Our contributions in this paper are twofold. First, we present an architecture and component design of a fuzzy inference-based scheme for CDN server selection, and a technique to choose some CDN system parameters properly to ensure resource utilization and user-perceived QoS satisfying design criteria and specifications. Second, we demonstrate the viability of the proposed scheme through simulation results and confirm that it offers superior performance over other traditional schemes. It is also shown that the proposed scheme incurs low overhead and is more scalable.

The remainder of this paper is organized as follows. In Section 2, we present a brief overview of dynamic CDN server selection, including a conceptual model, decision criteria, and related work. In Section 3, we present a fuzzy inference-based server selection scheme, including its system architecture, fuzzy rule-base, parameter selection, and performance analysis. An extensive performance evaluation is given in Section 4, which includes the simulation setting, system training, and performance comparisons with existing schemes. Section 5 offers further discussions on the proposed scheme and its extensions, and Section 6 gives concluding remarks and directions for future work.

2. Dynamic server selection in content distribution networks

In this section, we first formulate a conceptual role-based model for CDN server selection to facilitate our design and evaluation. We then discuss the decision criteria and their characteristics in practice that discourage a deterministic decision process. Related work is also reviewed.

2.1. CDN server selection

Fig. 1 depicts a role-based model of CDN server selection, where five roles are identified. CDN cache servers and their users are represented by S and U, respectively. R is a reference for users and is visible to a decision maker (D) or its prober (P). Probers collect decision inputs for the decision maker. R can be in the finest granularity, i.e. one R for each user in HTTP redirection, or in a much coarser granularity, e.g. a group of neighboring users sharing the same R in DNS resolution. Probers and decision maker are responsible for getting server and network status, locating user requests, and

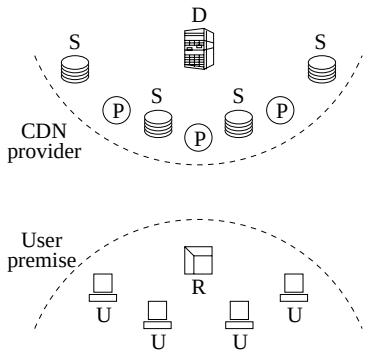

Fig. 1. A role-based model for server selection in content distribution networks.

determining which server will be used to serve a particular user request. In practice, probers, decision maker and servers are owned and maintained by the CDN provider, while users and R are in user premise and not under the control of any CDN provider.

2.2. Server selection criteria

To make a selection, the decision maker should know the location of a user and the latest network and server status. In the Internet context, a decision maker can only obtain the IP address of R by extracting the source address from the packets sent by R. To infer the location of R, the decision maker has to rely on other assistance such as reverse DNS resolution, address ownership lookup, Autonomous System (AS) number, or address prefix in Border Gateway Protocol (BGP) routing table. The host names in country-code top-level domain (ccTLD) usually can give a good hint on their geographical location, but this approach is less effective for names in generic top-level domain (gTLD). Moreover, reverse resolution often fails because either the database is unmaintained or inaccessible for external queries. IP address allocation authorities (i.e. ARIN, RIPE and APNIC) offer ownership record lookup services (e.g. WHOIS) for allocated addresses. However, for many large ISPs and organizations, a single block of IP addresses can spread throughout the Internet worldwide. AS number and BGP prefix usually can give better proximity in terms of network routing reachability. But even the longest BGP prefix is insufficient to infer user location and assess user–server proximity precisely. Although IP address allocation changes slowly when compared with network and server dynamics, any changes in network topology (e.g. when new links are added or existing links are decommissioned) may also change user location in network space.

On the other hand, the decision maker or prober has to measure individual network path and cache server periodically, and then to make prediction on network and server status. Since there are thousands of cache servers distributed across the Internet, it is impractical to have a very frequent measurement per server; otherwise, these measurements impose considerable network and server overhead (i.e. the measured or predicted server status known to the decision maker actually is not the current status at individual servers). Similarly, it is impractical to probe for network connectivity from all cache servers to a particular user. Since network status changes quickly, such measurements have to be done regularly. Therefore, only a small number of probers (compared with the large number of servers) can measure the latest network connectivity.

Synthesizing network status and server status that are measured separately with different metrics is another challenge. Usually, network connectivity is measured in terms of available bandwidth, hop count, and *rtt*; server status is measured in terms of available processor, memory, and I/O capacity. These two sets of measurement metrics are not fully compatible to each other. Moreover, the server selection made by CDN should be reasonably satisfactory from user's viewpoint. In CDN, user-perceived quality of experience can be highly characterized by the *click-to-display* latency, i.e. the time that users have to wait for a requested object to complete download from the server and to appear in web browser.

2.3. Related work

CDN and Web caching are two active research topics in recent years [1,4]. In this subsection, we focus on location estimation and server selection schemes, and refer to some surveys and the references therein on other related techniques such as active or passive Internet measurement [4].

Locating or clustering a user identified by an IP address in geographical or network space is of great interest to many service providers. For instance, an e-business application can customize itself for customers from different countries with different currencies. The proposed DNS LOC [5] record can specify the geographical location in latitude and longitude of an IP address. But normally it requires manual configuration and is more appropriate for servers instead of users. There are several commercial and proprietary geographical mapping services available, which are based on the information from reverse DNS resolution, address allocation lookup, and ISP address survey; however, nowadays they still cannot provide a satisfactory mapping success rate and accuracy.

Locating an IP address in network space is even more challenging. An alternative strategy is to cluster IP addresses [6,7] with respect to their AS number, BGP prefix, or local DNS server. For instance, IP addresses sharing the longest BGP prefix suggest strong network proximity. But unless there is one cache server in each cluster, this approach cannot estimate the inter-cluster proximity properly. It is also possible to assess the proximity of two IP addresses by correlating router/AS path or round-trip time to these two addresses from a probing point; however, the correlation is found not to be strong enough to make a consistent assessment over the heterogeneous Internet.

Global Networking Positioning (GNP) [8] is an attempt to use multiple reference points to locate an IP address in network space. It adopts a coordinate-based approach similar to that used in GPS. GNP first locates a few known landmarks, and then locates a given IP address by its relative distances to these landmarks. With the coordinates of two IP addresses, the network distance between them can be derived. But without the geometry triangulation like the one in GPS, GNP unavoidably requires a higher dimensional space or more landmarks, which involves higher computational overhead.

Internet Distance Map (IDMap) [9] is another attempt to provide network distance service between any two IP addresses with the assistance of tracers. Tracers measure the distance among them in a proactive manner. When a source wants to obtain the network distance to a destination, the source first contacts a nearby tracer; then IDMap locates the nearest tracer for the destination. With the inter-tracer distance information maintained by IDMap, a measurement of distance between the source or destination to the nearest tracer can be used to derive the source–destination distance accordingly. IDMap requires an infrastructure of tracers, which is not available yet. IDMap also suffers directional relative errors, e.g. a direct network path from source to destination may be shorter than that to any tracer.

Some server selection schemes are developed for users instead of CDN providers (see [10] and the references therein), and they usually consider hop count [11] or *rtt* [12]. SPAND [13] is located near users and passively monitors user-perceived performance. The measured data will be used for future server selections. The approach is inadequate for CDN since CDN providers have no access to user premise; i.e. even passive monitoring is infeasible. Application or IPv6 anycast is another approach that shifts the responsibility for server selection from users. In application anycast [14], cache servers share the same anycast domain name (ADN). An additional local ADN server is required to determine the performance metrics associated with a list of servers sharing the same ADN. The query processes are similar to those in the hierarchical DNS system; i.e. the local ADN server chooses an appropriate server from the server list. However, both application and IPv6 anycast schemes require a major infrastructure upgrade, as well as the extra resolve servers or routers that are close to users. On the contrary, a good CDN server selection scheme should be able to transparently and dynamically redirect user requests, and it should be easily deployed and efficiently maintained by CDN providers, not end users.

Other sophisticated server selection schemes, e.g. those employed by Cisco Distributed Director [3] (DD), can make decisions according to multiple metrics. DD probes server load and calculates AS hops between users and servers. Static weights are assigned to each metric. However, with such a static configuration, it is very difficult to efficiently utilize network resources in a highly dynamic network. There are other server or peer selection schemes. For example, [15] adopts a decision tree-based approach to gradually switching among different peers and finally settling down with the one considered best, when users exchange large objects in peer-to-peer file-sharing systems. However, in our context, a user request is atomic and only short-lived (e.g. requesting an HTML page), and CDN providers have to assign the request to a server proactively without switching servers.

3. A fuzzy inference-based server selection scheme

In this section, we first present the architecture and component design of the proposed fuzzy inference-based server selection scheme. We then focus on the inference rule-base design, followed by a thorough analysis on how to choose system parameters and derive performance bounds. Table 1 lists some symbols used in the rest of this paper and their descriptions, respectively.

3.1. Fuzzy inference system design

Fig. 2 illustrates an example CDN system, where cache servers (S) are scattered in four timezones. Server load and network distance are two major decision inputs in this system. We study the aggregated user-perceived performance represented by one reference point R; i.e. all users that

Table 1
Notations

Symbol	Description
rtt_i	Measured round trip time from prober i to a given user
sld_n	Advertised workload of a CDN cache server n
l_n	Likelihood of a request being served by server n
μ_X	Degree of being in a fuzzy set X
U_{r_i}	Fuzzy term set of rtt_i, $i=1,2,\ldots,M$ when there are M probers
U_{s_n}	Fuzzy term set of sld_n
U_r	Input fuzzy term set of the fuzzy inference engine
U_l	Fuzzy term set of l_n
R_{ik}	Fuzzy set in U_{r_i} of the kth fuzzy inference rule
S_k	Fuzzy set in U_{s_n} of the kth fuzzy inference rule
L_k	Fuzzy set in U_l of the kth fuzzy inference rule
I_{ik}	Input region for rtt_i of the kth rule in the fuzzy inference engine
I_k	Input region for sld_n of the kth rule in the fuzzy inference engine
O_k	Output region of the kth rule in the fuzzy inference engine
Q_k	Weight degree assigned to the kth rule in the fuzzy inference engine
U_t	Server utilization
L	Network load
N_s	Number of total CDN servers
P_B	User request blocking rate
P_D	User request dropping rate
t_d	User *click-to-display* latency
t_s	The time duration of a request staying in the serving server
B_s	Number of logical servers (e.g. httpd processes) in a physical server
μ	Service rate of each CDN server
Λ	Total user request rate
ρ	Total traffic intensity, $\rho=\lambda/N_s\mu$
P_I	Server load probing interval

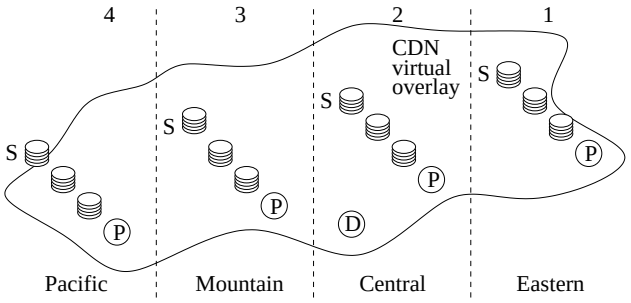

Fig. 2. A CDN system spreading across four timezones.

share the same R will be treated as an aggregated U. In each timezone, there is a limited number of probers (P) that actively measure their network distance in *rtt* to requesting users. Probers also periodically probe server load, in terms of number of active requests. The measured network and server status will be made available to a decision maker (D) that is driven by a fuzzy inference engine. Once a decision for a user request is made by the decision maker, the server assignment will be conveyed back to users.

Here, CDN servers only report the number of active requests that they have, not when these requests will be completed. This is due to the fact that many requests involve dynamic objects and their sizes cannot be determined beforehand. Also, many OS-related overheads such as task scheduling, memory paging, and I/O access can add extra uncertainty. In addition, the decision maker does not have the latest status on all servers, since servers only report their status periodically. Therefore, only the number of active requests reported by a server in the last report is available to the decision maker. Similarly, not all servers can have the network distance measurement to a requesting user. Due to the limited number of active measurements for a request, the measured *rtt* may contain considerable noise caused by network dynamics. Moreover, some users may be inaccessible for external probings, e.g. blocked by firewalls. In this case, P either provides no measurement for this requesting user, or only gives the best effort estimate such as a partial *rtt* if the probe is bounced by firewall. Therefore, only a small number of *rtts* to a requesting user are available to the decision maker.

In a deterministic selection system, the decision maker has to answer 'for a particular request, which server is chosen?' To this end, the decision maker has to quantize the measured server load and network distance precisely and to choose a server with the least load or shortest distance numerically. These measurements may be inaccurate and behind the actual changes. This is one of the main reasons that a deterministic system cannot perform properly with these constraints. In addition, a least loaded server may not have the shortest distance, and vice versa.

On the contrary, in a fuzzy inference-based system, the decision maker only needs to answer 'for a particular request, what is the likelihood l_i for server S_i to serve this request?' Assume there are N servers under consideration

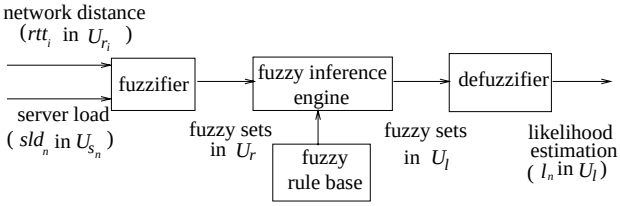

Fig. 3. A fuzzy inference-based system for CDN server selection.

and at most M probers available. The fuzzy system takes an $(N+M)$-dimension vector input of network and server status measurements and produces an N-dimension vector output of server assignment likelihood. In our design, we consider an $(M+1)$-input-1-output system for each server, since the design is generic and each system can be trained independently. Fig. 3 shows the block diagram of the proposed fuzzy inference system. For server n, its server load and M network distance measurements are first fuzzified with linguistic variables in fuzzy set U_r, which is the input term set of the fuzzy inference engine. For each measurement, unlike being treated quantitatively in a deterministic system, it is mapped to the fuzzy set U_r with a certain degree. Through a fuzzy inference engine driven by inference rules, the scalar output is mapped to another fuzzy set U_l, which is the term set of server likelihood. The defuzzifier eventually converts the qualitative assessment into a quantitative decision.

3.2. Inference rule base

The designed system employs a knowledge base, consisting of trained fuzzy inference rules, and an appropriate inference engine, to estimate the likelihood of a request that is served by a specific server, according to the measured network distance and server load. The inference system is capable of utilizing knowledge elicited from human operators, which is expressed in natural language, and its cardinal element is a linguistic variable [16]. Network distance and server load are two types of inputs of the proposed inference engine. Let linguistic variable rtt_i, $i=1,2,\ldots,M$, represent the *rtt* measurement from prober P_i to a given user; then the corresponding universe of discourse is the set of all possible *rtt* levels. We choose the term set of rtt_i, denoted by U_{r_i}, to contain the following elements: *very close* (VC), *close* (C), *not faraway* (NF), *faraway* (F), and *very faraway* (VF). Let linguistic variable sld_n represent the server load measurement of server n; then the corresponding universe is the set of all possible server load (*sld*) levels. We choose the term set of sld_n, denoted by U_{sn}, to be the set containing the following elements: *slightly loaded* (SL), *lightly loaded* (LL), *loaded* (L), *heavily loaded* (HL), and *extremely loaded* (EL).

On the other hand, server likelihood is the output of the proposed inference engine. Let linguistic variable l_n be the likelihood of a request served by server n with the range of [0,1]. We choose the term set of l_n, denoted by U_l, to be

the set containing the following element: *very unlikely* (VU), *unlikely* (UL), *uncertain* (UC), *likely* (L), and *very likely* (VL). The numbers of total terms in U_{ri}, U_{sn}, and U_l are chosen carefully to achieve a balance between complexity and performance. The membership functions of the fuzzy inference inputs (*M rtt and 1 sld* measurements) and output (server likelihood) depend on the actual network and server dynamics.

In the fuzzifier shown in Fig. 3, each specific value of the measured rtt_i is mapped to fuzzy set $U_{r_i}^1$ with degree $\mu_{x_i}^1(rtt_i)$, to fuzzy set $U_{r_i}^2$ with degree $\mu_{x_i}^2(rtt_i)$, and so on, where $U_{r_i}^J$ is the name of the Jth term or fuzzy set value in U_{r_i}. Each specific value of the measured server load sld_n is mapped to fuzzy set $U_{s_n}^1$ with degree $\mu_y^1(sld_n)$, to fuzzy set $U_{s_n}^2$ with degree $\mu_y^2(sld_n)$, and so on, where $U_{s_n}^L$ is the name of the Lth term or fuzzy set value in U_{s_n}.

The fuzzy rules describe the fuzzy-logic relationship between network and server measurements and server likelihood. The kth rule has the following format:

R_k: IF (rtt_0 is R_{0k}) and … and (rtt_M is R_{Mk}) and (sld_n is S_k) THEN (l_n is L_k),

where $k=1,2,\ldots,K$, and K is the number of rules, $(rtt_0, \ldots, rtt_M, sld_n) \in U_{r_0} \times \cdots \times U_{r_M} \times U_{s_n} = U_r$ and $l_n \in U_{l_n} = U_l$ are linguistic variables, R_{ik}, S_k and L_k are fuzzy sets in U_{r_i}, U_{S_n} and U_{l_n} respectively.

In the fuzzy inference engine, the product inference engine is employed to combine the fuzzy rules into a mapping from fuzzy sets in U_r to fuzzy set in U_l, or

Given Fact: (rtt_0 is $\tilde{R}_0$) and … and (rtt_M is $\tilde{R}_M$) and (sld_n is $\tilde{S}_n$)
Consequence: (l_n is $\tilde{L}_n$),

where $\tilde{R}_i$, $\tilde{S}_n$ and $\tilde{L}_n$ are linguistic terms for rtt_i, sld_n and l_n, respectively.

The fuzzy rule-base is created by a training data set that consists of measured input–output pairs. To avoid tedious field trials, the training data can be generated by computer. The table-lookup approach is employed to generate IF–THEN rules. The degree assigned to rule k, denoted by Q_k, is calculated by using the following product operation

$$Q_k = \mu_k \prod_{i=0}^{M} \mu_{I_{ik}}(rtt_i)\mu_{I_k}(sld_n)\mu_{O_k}(l_n), \tag{1}$$

where I_{ik} denotes the input region of rule k for rtt_i, I_k denotes the input region for sld_n, O_k denotes the output region for l_n, $\mu_{I_{ik}}(rtt_i)$ is the degree of rtt_i in I_{ik} obtained from the membership functions, $\mu_{I_k}(sld_n)$ is the degree of sld_n in I_k, $\mu_{O_k}(l_n)$ is the degree of l_n in O_k, and μ_k, which is between 0 and 1, is the degree assigned by human operators.

The defuzzifier performs a mapping from fuzzy set $L_n \in U_l$ to a crisp point l_n. Among the commonly used strategies, the center average defuzzification method yields a superior result [17]. Let $\tilde{l}_n$ denote the estimation (generated by the fuzzy inference system at time t_n) of the likelihood l_n. The formula for the estimation at the defuzzifier output is

$$\tilde{l}_n = \frac{\sum_{k=1}^{K} \bar{Q}_k \prod_{j=1}^{M} \mu_{I_{jk}}(rtt_j)\mu_{I_k}(sld_n)\bar{l}_k}{\sum_{k=1}^{K} \bar{Q}_k \prod_{j=1}^{M} \mu_{I_{jk}}(rtt_j)\mu_{I_k}(sld_n)}, \tag{2}$$

where $\bar{l}_k$ is the center output region value of rule k, and $\bar{Q}_k$ is the degree (normalized to 1) of rule k.

The robustness of fuzzy inference systems and approaches to designing stable fuzzy control systems have been discussed in [18]. Stability of closed-loop systems with fuzzy logic controllers has been investigated in [19]. In general, robust fuzzy control is an open problem. The potential performance oscillations can be reduced or eliminated if the probing frequency is relatively high when compared with the system dynamics, e.g. the variation of request arrival rate.

3.3. System performance analysis

3.3.1. Selecting system parameters

It is important to choose system parameters properly to ensure system performance and user-perceived QoS satisfying design criteria and specifications. Here, system efficiency is measured by server utilization (U_t) and network load (L). In CDN, N_s servers are dedicated for service provisioning. From service provider's viewpoint, it is desirable to maximize U_t and minimize N_s. L is the product of bandwidth consumed by and distance traveled by conveyed traffic. From network provider's viewpoint, serving a request at the nearest server can minimize L. On the other hand, user-perceived QoS is measured in request blocking rate (P_B), average *click-to-display* latency (t_d), and request dropping rate (P_D). In CDN, each cache server (S) has a known service rate (μ) and a configurable number (B_s) of maximum logical servers (e.g. httpd processes). If a server already has B_s active requests, the newly arrived request will be *blocked* (we omit a small listen( ) queue in TCP/IP protocol stack). When experiencing an excessive latency, users may cancel their requests (e.g. pressing the stop button in their browsers), and it is counted as a request *dropping* event.

Given the total request volume (Λ) and service rate (μ) of a server, a system designer needs to make the following decisions: (i) how many servers (N_s) deployed? (ii) how many server processes (B_s) supported per physical server (we use httpd process as an example)? (iii) how exactly requests being served? iv) where to locate these servers? The last question should take both network topology and request pattern into account, and is beyond the scope of this paper. For (iii), there are two possible serving schemes. One is a *serial* scheme: the requests are buffered in a *virtual* queue and served with FIFO discipline. Once a request is being served, its httpd process gets the full service rate of μ. The other is a *parallel* scheme: if there are $k(1 \le k \le B_s)$

active httpd processes, each of them can get a service rate of μ/k equally. Both the serial and parallel schemes have their own pros and cons. Generally speaking, if the variance of the size distribution of requested objects is large, the parallel scheme has better performance in terms of average latency, and vice versa. Since a large percentage of web objects are text-only of a few kilobytes, while few objects contain multimedia content and have a large variance of size, an exponentially distributed object size is acceptable in this context. In Appendix A, we show that when the size of requested objects follows an exponential distribution, both parallel and serial schemes are equivalent in terms of request blocking rate and average latency. Here, we use the serial scheme with an exponential object size for analysis purpose. The results can be applied to the parallel scheme accordingly.

To obtain an optimal N_s and B_s, we formulate an optimization problem with the constraint of a bounded P_D, since a dropped request not only wastes shared system resources but also brings bad user experience. From the viewpoint of CDN providers and users, our goal is to minimize the cost (utility) function

$$C = c_1 N_s + c_2 E[t_d] + c_3 P_B. \tag{3}$$

Here, we assume that the cost to purchase and maintain cache servers is proportional to N_s. The c_is are the coefficients that link the CDN investment and user QoS measure.

A larger B_s can reduce P_B, but it also increases $E[t_d]$ and P_D when the request intensity ρ, i.e. $\Lambda/(\mu N_s)$, increases. To bound P_D, B_s must be bounded. When a request arrives at a server with q requests buffered in the queue, the probability of the new request staying in the server for more than t time units is

$$\Pr\{t_s > t | Q = q\} = \sum_{i=0}^{q-1} \frac{\exp(-\mu t)(\mu t)^i}{i!},$$

where t_s is the delay at the server. Let t_s^* be the maximum tolerable delay at the server. B_s can be set as the maximum q such that $\Pr\{T_s > T_s^* | Q = q\} < P_D^*$, where P_D^* is the maximum tolerable P_D. Since the service time for each request is i.i.d. with mean $1/\mu$ and variance $1/\mu^2$, the sum of service time for q requests can be approximated by a Gaussian random variable with mean q/μ and variance q/μ^2. Thus

$$B_s = \left\{ \max(q) : Q\left(\frac{T_s^* - q/\mu}{\sqrt{q}/\mu}\right) < P_D^*; q > 0 \right\}, \tag{4}$$

where $Q(\cdot)$ is defined as

$$Q(x) = \int_x^\infty \frac{1}{\sqrt{2x}} \exp\left(-\frac{y^{\sqrt{2\pi}}}{2}\right) \mathrm{d}y.$$

Since the user population is large and the probability of a user requesting an object in a small interval is small, the aggregated request traffic can be modeled as a Poisson process with rate Λ. The blocking probability P_B can be calculated as

$$P_B = \frac{\rho^{B_s}(1-\rho)}{1-\rho^{B_s+1}}. \tag{5}$$

The average time for a request staying with a server, including queuing delay and processing time, is

$$E[t_s] = \left(\frac{1}{1-\rho} + \frac{B_s \rho^{B_s}}{1-\rho^{B_s}}\right) \Big/ \mu. \tag{6}$$

Hence, the average user experienced latency, $E[t_d]$, is

$$E[t_d] = \left(\frac{1}{1-\rho} + \frac{B_s \rho^{B_s}}{1-\rho^{B_s}}\right) \Big/ \mu + xE[rtt], \tag{7}$$

where $xE[rtt]$ is x rounds of network-related overhead in rtt for a request. Here, we adopt $x=2$ by taking into account both TCP connection establishment and HTTP request–reply transaction. Substituting (5) and (7) into (3), we can determine the optimal N_s for a CDN system.

3.3.2. Deriving performance benchmark

We derive system performance bounds, which can be used as a benchmark to compare the performance of different schemes, and to identify the performance margin for future improvement. The benchmark scheme randomly distributes arrival requests to all servers uniformly. With this scheme, the minimum blocking rate $(P_B)_{\min}$, minimum average latency at server $(E[t_s])_{\min}$, and maximum server utilization $(U_t)_{\max}$ are

$$(P_B)_{\min} = \frac{(1-\rho)\rho^{B_s}}{1-\rho^{B_s+1}}, \tag{8}$$

$$(E[t_s])_{\min} = \frac{\rho(1-\rho^{N_s} - N_s\rho^{N_s}(1-\rho))}{\mu(1-\rho)}, \tag{9}$$

$$(U_t)_{\max} = \frac{(1-\rho^{N_s})\rho}{1-\rho^{N_s+1}}. \tag{10}$$

The benchmark scheme (i.e. a random selection scheme) may choose a faraway server even when a closer server is available, which increases network load. Alternatively, we can select servers uniformly within the same timezone where requests originate. However, when the request pattern is uneven, the inzone scheme may block requests in a busy zone while servers in other zones are available. An ideal server selection scheme should choose server appropriately, so that P_B and $E[t_d]$ are close to those of the uniform scheme no matter whether the request pattern is even or uneven, and the network load is close to that of the inzone scheme if the request pattern is even.

4. Performance evaluation

4.1. Simulation topology and parameters

To compare and evaluate the performance of the proposed fuzzy inference-based server selection scheme with existing schemes, extensive simulations have been performed. The network topology used in our simulation is the same as that shown in Fig. 2. Twelve CDN cache servers (S) with the same service rate are equally located in four timezones, and there is one prober (P) in each zone. The probers collect the number of active httpd processes from all servers every P_I seconds. When a user initiates a request, the *rtt*s between all probers and this requesting user are measured.

Since users may be located anywhere, the network distance measured by *rtt* between user and server is randomly distributed in practice. In the simulation, if a user and a server are in the same zone, their *rtt* is truncated-normally distributed between 5 and 50 ms, with mean 10 ms and standard deviation 10 ms; if the user and the server are k ($k=1$–3) zones away, their *rtt* is truncated-normally distributed between $10k$ and $100k$ ms, with mean $50k$ ms and standard deviation $20k$ ms.

All servers have the same processing power and the size of requested objects is exponentially distributed, so the service time is exponentially distributed with a mean of 200 ms. Each server can have at most 18 active httpd processes, so that even when the traffic intensity is larger than 1 (i.e. each server always has a full queue), the dropping rate P_D will be less than 5% if user can tolerate at most 5 s latency, or P_D will be less than 0.3% if user can tolerate 6 s latency, according to our analysis in (4).

At some time instances, different zones can have different and uneven request patterns. For example, at 9 a.m. EST, the request arrival in Eastern timezone is much more intensive than that in Pacific timezone. To examine the system performance in a wide spectrum of network scenarios, simulations are performed with traffic intensity increasing from 10% to 90% for both even and uneven request patterns.

4.2. Simulated fuzzy system

Before presenting the simulation results, we first show how the fuzzy inference system is trained and tuned. After four probers are chosen, the designed system takes four *rtt* measurements from four probers and the load history at a specific server to estimate the likelihood that a request is served by this server.

For the type of membership functions, it is necessary to take into account both the computational efficiency and adaption easiness of the fuzzy inference system. Gaussian, triangular and trapezoidal functions are the most commonly used membership functions. Here, we choose the Gaussian membership function since it can better reflect the nature of the network and server load status in a CDN. With the Gaussian function, the degree $\mu_{I_{jk}}(rtt_j)$ and $\mu_{I_k}(sld_n)$ in (1) can be expressed as

$$\mu_{I_{jk}}(rtt_j) = \exp\left(-\left(\frac{rtt_j - \overline{rtt}_{jk}}{\sigma^r_{jk}}\right)^2\right), \tag{11}$$

$$\mu_{I_k}(sld_n) = \exp\left(-\left(\frac{sld_n - \overline{sld}_k}{\sigma^s_k}\right)^2\right), \tag{12}$$

where $\overline{rtt}_{jk}$, σ^r_{jk}, $\overline{sld}_k$, and σ^s_k are adjustable parameters for each Gaussian function.

Substituting (11) and (12) into (1), the estimate at the defuzzifier output is

$$\tilde{l}_n = \frac{\sum_k \bar{Q}_k \prod_j \exp\left(-\left(\frac{rtt_j-\overline{rtt}_{jk}}{\sigma^r_{jk}}\right)^2\right)\exp\left(-\left(\frac{sld_n-\overline{sld}_k}{\sigma^s_k}\right)^2\right)\bar{l}_k}{\sum_k \bar{Q}_k \prod_j \exp\left(-\left(\frac{rtt_j-\overline{rtt}_{jk}}{\sigma^r_{jk}}\right)^2\right)\exp\left(-\left(\frac{sld_n-\overline{sld}_k}{\sigma^s_k}\right)^2\right)}. \tag{13}$$

The initial center values for elements in $U_l(\bar{l}_k)$ are 0, 0.25, 0.50, 0.75, and 1.0, respectively. In order to determine $\overline{rtt}_{jk}$ and σ^r_{jk}, the possible *rtt*s are divided into five ranges, and the initial values of $\overline{rtt}_{jk}$ and σ^r_{jk} are determined based on the mean and variance of the *rtt* in each range, denoted, respectively, as $\overline{rtt}_{jk}(0)$ and $\sigma^r_{jk}(0)$. The initial values of $\overline{sld}_k$ and σ^s_k, denoted as $\overline{sld}_k(0)$ and $\sigma^s_k(0)$, respectively, are set in a similar way. To obtain the initial fuzzy inference rules, 10,000 requests are generated by computer. During training process, it is assumed that the object size and *rtt*s from user to all servers are readily available, so that the decision can be made based on the accurate *click-to-display* latency t_d. When there is a request, the decision of which server is selected is based on the value of t_ds for a user w.r.t. all servers. All t_ds are sorted and the likelihood values are assigned as follows: the server with the minimum t_d is assigned as VL; the servers with the second and third minimum t_ds are assigned as L; the servers with fourth to sixth minimum t_ds are assigned as UC; the servers with seventh to ninth minimum t_ds are assigned as UL, and the servers with the three maximum t_ds are assigned as VU. After the initial fuzzy inference rules have been generated, the total number of fuzzy rules K is known. In order to determine the optimal fuzzy inference rules, the back propagation training method, which is an iterative gradient algorithm, is employed to train the fuzzy system, i.e. given a set of training input–output sequences (rtt_j, sld_n, l_n), $j=1$–4, the parameters in (13) are adjusted so that the decision error at step m

$$\mathrm{err}(m) = \frac{1}{2}(\tilde{l}_n - l_n)^2 \tag{14}$$

can be minimized. Since $\tilde{l}_n$ is a function of $\overline{rtt}_{jk}$, σ^r_{jk}, $\overline{sld}_k$, σ^s_k and $\bar{l}_k$, the optimization problem becomes the one by training the parameters $\overline{rtt}_{jk}$, σ^r_{jk}, $\overline{sld}_k$, σ^s_k, and $\bar{l}_k$ to minimize

err(m). At each step, the gradient of err(m) with respect to the adjusted parameter is calculated by differentiating err(m) with respect to the concerned parameter, then the parameter is adjusted based on the gradient value.

Let

$$z_k = \prod_{j=1}^{4} \exp\left(-\left(\frac{rtt_j - \overline{rtt}_{jk}}{\sigma^r_{jk}}\right)^2\right) \exp\left(-\left(\frac{sld_n - \overline{sld}_k}{\sigma^r_k}\right)^2\right),$$

$$b = \sum_{k=1}^{K} z_k, \qquad c = \sum_{k=1}^{K} (\tilde{l}_k z_k), \quad \text{then } \tilde{l}_n = c/b.$$

To adjust $\tilde{l}_k$, we use

$$\tilde{l}_k(m) = \tilde{l}_k(m-1) - \alpha \frac{\partial \mathrm{err}(m)}{\partial \tilde{l}_k}, \tag{15}$$

where α is a positive real-valued constant step-size.

Using the chain rule, we have

$$\frac{\partial \mathrm{err}(m)}{\partial \tilde{l}_k} = (\tilde{l}_n - l_n) \frac{\partial \tilde{l}_n}{\partial c} \frac{\partial c}{\partial \tilde{l}_k} = (\tilde{l}_n - l_n) \frac{1}{b} z_k. \tag{16}$$

Hence, the algorithm to adjust $\bar{l}_k$ is

$$\bar{l}_k(m) = \bar{l}_k(m-1) - \alpha(\tilde{l}_n - l_n) \frac{1}{b} z_k, \tag{17}$$

where $n = 1,2,\ldots,N$. Similarly, we can obtain the algorithms to adjust $\overline{rtt}_{jk}$, σ^r_{jk}, $\overline{sld}_k$, and σ^s_k, where $n = 1,2,\ldots, j = 1, 2, 3, 4$, and $k = 1,2,\ldots,K$.

After the parameters of $\overline{rtt}_{jk}$, σ^r_{jk}, $\overline{sld}_k$, σ^s_k and $\bar{l}_k$ have been adjusted with these algorithms, the fuzzy inference rules can be further tuned according to the adjusted values of parameters and the same dataset which is used to generate the initial fuzzy inference rules.

Fig. 4(a) and (b) shows the final membership functions of rtt_i and sld_n, respectively. After the training process, the center values of the elements in U_l, $\bar{l}_k$, are set to 0, 0.233, 0.489, 0.762, and 1.0.

4.3. Performance comparison

The performance of the designed fuzzy inference scheme is compared with three other schemes: (i) *random*, randomly selecting from all servers with equal likelihood; (ii) *inzone*, randomly selecting from all servers in the zone where the prober has the least *rtt* to the user; (iii) *static setting*, randomly selecting from three servers with the least estimated response time, i.e. $sld_n \times E[t_s] + x \times rtt_n$, where x is the weight of *rtt*, sld_n is the advertised load of server n, and rtt_n is the measured *rtt* between the user and the prober that is in the same zone as server n. If x is too large, it is very likely to overload servers in congested zones when some faraway servers are idle; if x is too small, it is very likely to choose a faraway server while some nearby servers are just lightly loaded. x is set to 2 in our simulation. In addition, all schemes employ a certain degree of randomization to mitigate the oscillation problem.

The system performance comparison includes request blocking rate P_B and average network load L. Since L is proportional to the *rtt* for a given request, the average serving *rtt* is chosen to represent L. User-perceived QoS is measured by the average *click-to-display* latency $\bar{t}_d$ and request dropping rate P_D. Simulation results are presented in Section 4.3.1 when requests are evenly distributed across the network, and in Section 4.3.2 when requests are unevenly distributed in different zones, respectively. The measurement overheads of different schemes are compared in Section 4.3.4.

4.3.1. Even request pattern

When requests are evenly generated in the network, for the random and inzone schemes, the request arrival rates for each server are similar. Therefore, their blocking rates, P_Bs, should be similar, and the schemes are also the optimal schemes in terms of blocking rate when the decision maker has no status information of each server. Fig. 5 shows P_B (in y-axis) for all schemes when the traffic intensity ρ (x-axis) is

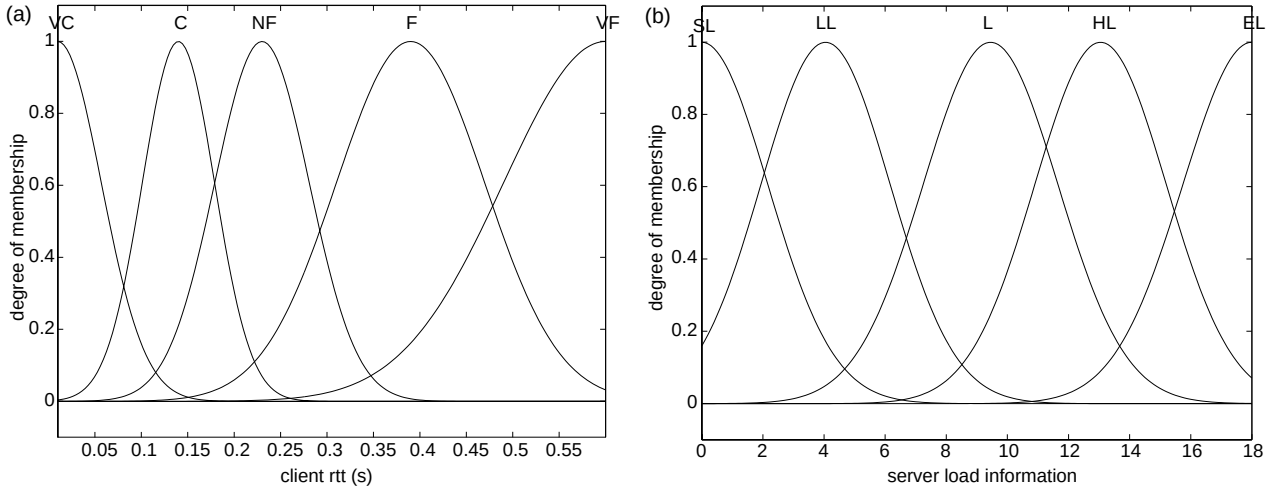

Fig. 4. Membership functions of (a) *rtt* and (b) *sld* after training.

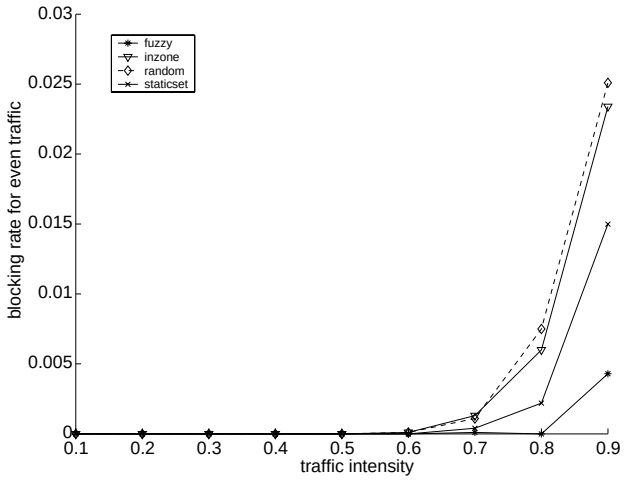

Fig. 5. Blocking rate with even request pattern ($P_I = 1$ s).

increasing from 0.1 to 0.9. In the simulation, the probers update server status every $P_I = 1$ s. The figure shows that all schemes have no request blocking until $\rho = 0.7$. When $\rho = 0.9$, the blocking rates for the random and inzone schemes are around 2.5%, and the blocking for the static setting and fuzzy schemes are around 1.5% and 0.5%, respectively. With the same historical server load information and partial *rtt* measurements, the blocking rate for the fuzzy scheme is much lower than that for the static setting scheme, since the fuzzy system can intelligently give a higher priority to server load metric when the arrival traffic becomes heavier.

Fig. 6 shows the average network load (in terms of *rtt*), which is normalized to the load of the random scheme, w.r.t. traffic intensity. The average network loads of the random and inzone schemes remain constant, which give the upper and lower bounds, respectively. The network loads of the static setting and fuzzy schemes approach the lower bound when the traffic intensity is low, and increase when there are more requests, which is a desired behavior. When requests

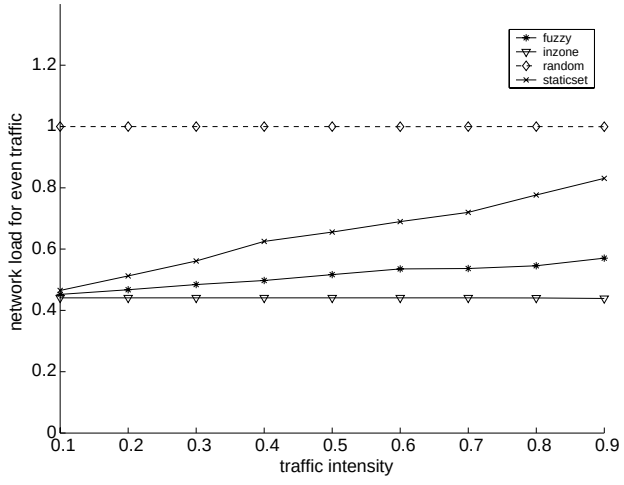

Fig. 6. Normalized average network load with even request pattern ($P_I = 1$ s).

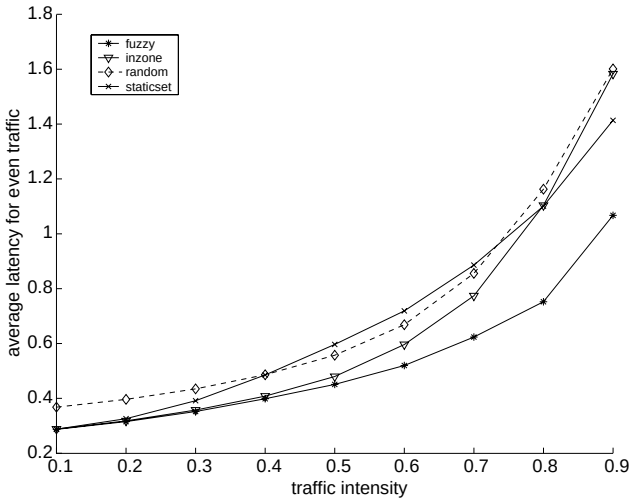

Fig. 7. Average latency with even request pattern ($P_I = 1$ s).

become intensive, they make a trade-off between blocking rate and network load, by intentionally choosing some faraway servers when nearby servers are very busy. As shown in Fig. 6, the load of the fuzzy scheme is constantly smaller than that of the static setting scheme. Since the zone information of the user is unavailable, the decision maker has to estimate the user location based on the measured *rtt* between the user and all probers. The estimation may not be accurate since, in network space, a user may be quite close to one prober, but not that close to any server in the same zone. Since the fuzzy scheme is intrinsically tolerable to measurement noise and errors, it can achieve a lower blocking rate and smaller network load than the static scheme simultaneously.

Fig. 7 shows the average of user perceived latency in seconds. User perceived latency has two components: the *rtt* between the user and the server, and the delay at the server. When the traffic intensity is light, the latency is dominated by *rtt*; when requests become intensive, the latency is dominated by the delay at the server. Therefore, the average latency for the inzone scheme is the optimal one when the request intensity is small, but it increases faster than the random scheme when there are more requests, and they meet when $\rho = 0.9$. The performance of the fuzzy scheme is very respectful, which approaches the lower bound when the traffic intensity is light, and the latency increases much slower than all the other schemes and remains the lowest all time. In other words, the fuzzy scheme can provide better QoS no matter whether the traffic intensity is low or high.

4.3.2. Uneven request pattern

In this set of simulations, the request arrival ratio from Eastern (zone 1) to Pacific timezone (zone 4) is 0.4:0.3:0.2:0.1. Under this scenario, the performance of the random scheme is rarely affected. However, as shown in Fig. 8, blocking occurs for the inzone scheme when $\rho \geq 0.6$, and its blocking rate is around 9% when $\rho = 0.9$. Since the traffic intensity in Eastern timezone equals $0.4\rho/0.25$, it exceeds 1 when $\rho \geq 0.625$, and

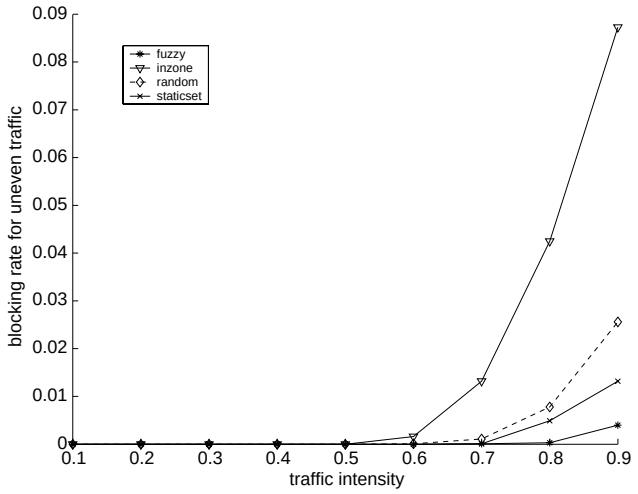

Fig. 8. Blocking rate with uneven request pattern ($P_I = 1$ s).

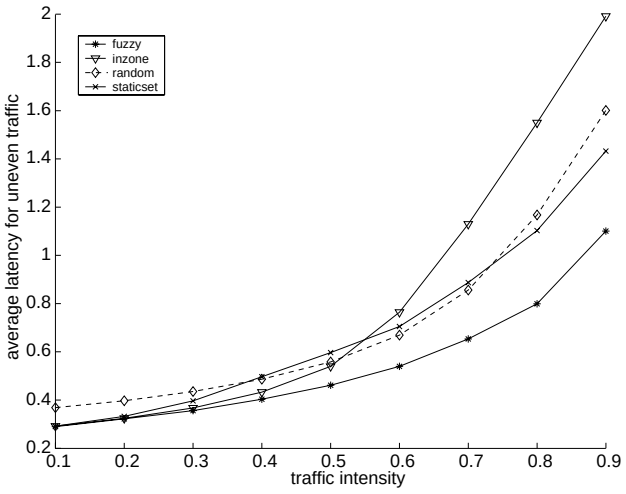

Fig. 10. Average latency with uneven request pattern ($P_I = 1$ s).

equals 1.6 when $\rho = 0.9$. Similarly, the traffic intensity in Central timezone reaches 1 when $\rho = 5/6$. Therefore, it is not surprising to see the severe blocking rate with uneven requests for the inzone scheme as shown Fig. 8. Also, the average latency of the inzone scheme increases quickly and exceeds that of the random scheme when $\rho \geq 0.6$, Fig. 10. The network loads for the inzone and random schemes remain almost unchanged, as shown in Fig. 9, since they do not consider the dynamics of arrival traffic when making decisions. On the other hand, the fuzzy scheme still has the least blocking rate and average latency among all schemes. Its average network load slightly increases since it directs some requests from eastern zones to the servers in western zones to alleviate the congestion. Therefore, the fuzzy scheme is very robust with different traffic patterns.

4.3.3. Dropping rate

When $B_s = 18$ and maximum tolerable latency is 5 s, the request dropping rate for all schemes are negligible for an even request pattern: there is no dropping for the fuzzy scheme; the other three schemes have less than 0.5% dropping when $\rho = 0.9$. For uneven scenarios, the dropping rate for inzone scheme is the highest: around 1% unblocked requests dropped when $\rho = 0.9$ since the server load in eastern zones exceed 1 with the inzone scheme. Nevertheless, all dropping rates are less than the designed bound derived in Section 4.1. In other words, the dropping rate is guaranteed when B_s is appropriately chosen according to our analysis in (4).

4.3.4. Measurement overhead comparison

Both the fuzzy and static setting schemes require server load information which is probed periodically. The frequent probing itself may produce a considerable amount of extra traffic which is the overhead of these schemes. On the other hand, the larger the probing interval P_I is, the less accurate the load information is; therefore, it is quite difficult to select a server appropriately. Fig. 11 shows the blocking rate of these two schemes when the probing intervals are 1 and 2 s, respectively. With the probing frequency halved to 0.5

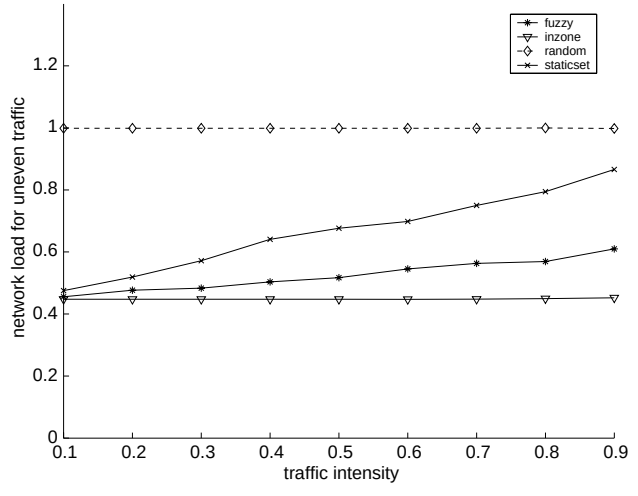

Fig. 9. Normalized average network load with uneven request pattern ($P_I = 1$ s).

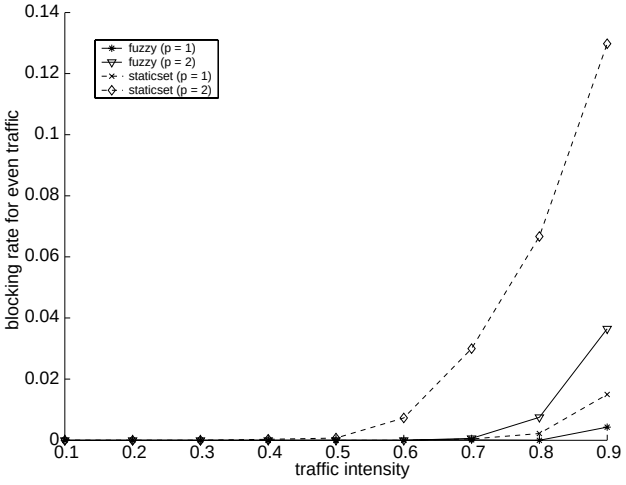

Fig. 11. Blocking rate for $P_I = 1$ s vs. $P_I = 2$ s (even request).

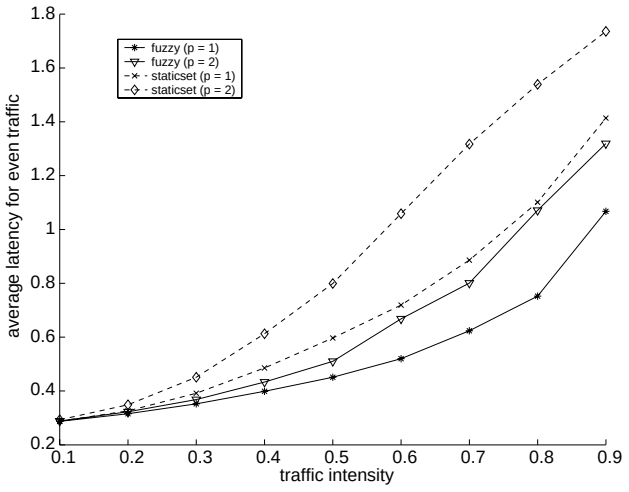

Fig. 12. Average latency for $P_I=1$ s vs. $P_I=2$ s (even request).

per second, the blocking becomes very severe for the static setting scheme, which blocks 13% of the requests when $\rho=0.9$. For the fuzzy scheme with the same probing frequency, the blocking rate is only 3.6% when $\rho=0.9$. Since in our simulated system, each server can serve 10 requests in 2 s on the average, the server load information 2 s ago may be drastically different from the current situation. The static scheme fails to deal with such error-prone inputs, while the fuzzy scheme can perform reasonably well. Fig. 12 shows the average latency of the fuzzy scheme is still much smaller than that of the static setting scheme. The network loads with these two schemes are almost unchanged, which is not presented here due to the limited space. In summary, the fuzzy scheme is quite robust with inaccurate load information and can tolerate a larger P_I.

5. Further discussions

Through the analysis in Section 3 and evaluation in Section 4, we have shown that the deficiencies of the deterministic schemes for CDN server selection are due to the fact that they strongly rely on the measurement inputs, which, however, are inaccurate and inconsistent in reality. Also, the static setting cannot deal with network dynamics efficiently. The proposed fuzzy inference-based scheme, as demonstrated in Section 4.3, is intrinsically robust and efficient with multiple, inaccurate, and inconsistent measurement inputs, and is adaptive to different network scenarios. This fact suggests that fuzzy inference is a viable approach to improve the system performance and user-perceived QoS in existing CDN systems. Instead of pushing the envelope of measurement accuracy on server load and network connectivity, alternatively, we can adopt a more adaptive and responsive fuzzy inference-based server selection scheme. Moreover, the fuzzy inference-based scheme imposes less measurement overhead (lower P_I) while achieving satisfactory performance.

Although our simulation is based on a simplified CDN system and parameter set, these issues only affect the training process of the fuzzy rule-base, and the fuzzy inference-based scheme still has its intrinsic advantages. With a properly designed membership function, the fuzzy scheme can achieve better performances on any general network topology and server placement. Since our system considers both server load and network connectivity, it can avoid the embarrassing situation (e.g. a least loaded server is unreachable or a nearest server is overloaded) where only server or network status is considered. Due to its embedded simplicity, the fuzzy scheme can take more inputs (e.g. user location) into account to handle more network and server dynamics, e.g. link up or down, request surge, etc.

The proposed scheme is also scalable to a very large CDN system. It only requires a small number of probers, and only collects server status at a lower frequency. It can also work in a hierarchical server selection system. For example, with regard to user location in network space, the proposed scheme can first identify one or a few appropriate server groups by one decision maker with a coarse membership function, and then zoom in a specific server by another decision maker with a fine-grained inference rule-base. This strategy not only further reduces the computational complexity, but also allows a certain degree of randomization among servers that are within a group and appear similar to requesting users. This feature can avoid the system oscillation in the case when a server is known to be the least loaded a moment ago will be soon overwhelmed by all requests redirected by a deterministic scheme.

6. Conclusions

In this paper, we have presented a fuzzy inference-based scheme for CDN server selection. By appropriately interpreting partial measurements of network connectivity and historical information of server load, the scheme can achieve higher resource utilization and provide better user-perceived QoS with less measurement overhead, due to its intrinsic capability of dealing with multiple, inaccurate, and inconsistent decision inputs. For future research, we intend to develop a hierarchical inference engine with multiple cooperative decision makers. We are also interested in introducing other metrics such as pricing for inter-CDN server selection when a service is offered by multiple service and delivery providers. Given the demonstrated functionality and performance advantages, we will continue to explore the fuzzy inference-based approaches in these contexts.

Appendix A. Serial vs. parallel serving schemes

Without loss of generality, let a physical server S have a fixed capacity r bps. The size of requested objects is exponentially distributed with mean m bits. The arrival of

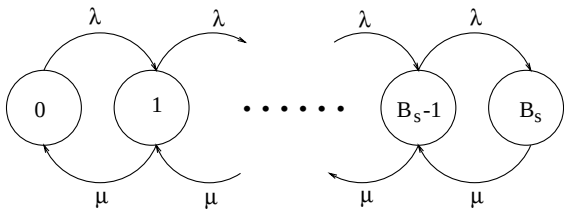

Fig. A1. M/M/1/B_s Markov chain.

requests is a Poisson process with the arrival rate λ requests per second. There are at most B_s requests being served simultaneously by S. The system is in state i if the number of active requests in S is i. The steady state probability of system in state i is denoted by p_i. Denote t_s as the time a request staying with the server, i.e. the time duration from a request arriving at the server to the whole requested object being transmitted.

A.1. Serial scheme

The serial system is a typical M/M/1/B_s queuing system, as shown in Fig. A1. In a nonzero state, the service rate, μ, equals r/m. The state transition rate from $i-1$ to i is λ, and that from i to $i-1$ is μ, where $i=1,2,\ldots,B_s$. The blocking probability, P_B, equals $p_{B_s} = ((1-\rho)\rho^{B_s})/(1-\rho^{B_s+1})$, where $\rho=\lambda/\mu$.

In state i, the average delay of a request is i/μ. Therefore, the mean of t_s in the serial system is

$$E[t_s] = \sum_{i=1}^{B_s} \frac{p_i i}{(1-p_0)r/m}. \tag{A.1}$$

A.2. Parallel scheme

In the parallel system, the state transitions can also be described by an Markov chain with arrival rate λ.

In state i, there are i httpd processes and each of them has a service rate $\mu_i=\mu/i=r/(m\times i)$, where $i=1,2,\ldots,B_s$. Since the size of requested objects satisfies (memoryless) exponential distribution, at any time instance, the size of the remaining unserved objects still satisfies the same exponential distribution. Therefore, the expected service time of each remaining object is exponentially distributed with mean $m\times i/r$. Among i remaining objects which are i.i.d., the service time of the smallest one is exponentially distributed with mean $(m\times i/r)/i = m/r$. Thus, the state transition rate from i to $i-1$ is $r/m=\mu$. Therefore, the Markov chain for the parallel system is exactly the same as that for the serial system, as shown in Fig. A1, and the steady state probability p_i and blocking probability $P_B = p_{B_s}$ are the same for both systems.

Since t_s equals the inverse of service rate of an httpd process, the expected value of t_s can be calculated as follows

$$E[t_s] = E[1/\mu'] = \frac{1}{1-p_0}\sum_{i=1}^{B_s}\frac{p_i}{\mu_i} = \sum_{i=1}^{B_s}\frac{p_i i}{(1-p_0)r/m}, \tag{A.2}$$

where μ' is the service rate of an httpd process. Comparing (A.1) and (A.2), both systems have the same average t_s.

References

[1] G. Barish, K. Obraczka, World wide web caching: trends and techniques, IEEE Commun. Mag. 38 (5) (2000) 178–184.
[2] J. Pan, Y.T. Hou, B. Li, An overview of DNS-based server selections in content distribution networks, Comput. Networks 43 (2003) 695–711.
[3] Cisco Distributed Director, http://www.cisco.com/univercd/cc/td/doc/product/iaabu/distrdir/.
[4] Cooperative Association for Internet Data Analysis (CAIDA), Internet tools taxonomy, http://www.caida.org/tools/taxonomy/.
[5] C. Davis, P. Vixie, T. Goodwin, I. Dickinson, A means for expressing location information in the domain name system, IETF RFC 1876 (1996).
[6] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, F. Zane, Clustering and server selection using passive monitoring, IEEE INFOCOM'02 (2002).
[7] B. Krishnamurthy, On network-aware clustering of web clients, ACM SIGCOMM'2000 (2000) 97–110.
[8] T. Ng, H. Zhang, Predicting Internet network distance with coordinates-based approaches, IEEE INFOCOM'02 (2002).
[9] Y. Jin, D. Shavitt, S. Jamin, C. Jin, L. Zhang, On the placement of Internet instrumentation, IEEE INFOCOM'2000 (2000).
[10] S. Dykes, K. Robbins, C. Jeffery, An empirical evaluation of client-side server selection algorithms, IEEE INFOCOM'2000 (2000).
[11] J. Guyton, M. Schwartz, Locating nearby copies of replicated Internet servers, ACM SIGCOMM'95 (1995) 288–298.
[12] R. Carter, M. Crovella, Dynamic server selection using bandwidth probing in wide area networks, IEEE INFOCOM'97 (1997).
[13] M. Stemm, S. Seshan, R. Katz, A network measurement architecture for adaptive applications, IEEE INFOCOM'01 (2001).
[14] E. Zegura, M. Ammar, Z. Fei, S. Bhattacharjee, Application-layer anycasting: a server selection architecture and use in a replicated Web service, IEEE/ACM Trans. Network 8 (4) (2000) 455–466.
[15] D. Bernstein, Z. Feng, B. Levine, S. Zilberstein, Adaptive peer selection, Proc. IPTPS'03 (2003).
[16] X. Shen, J.W. Mark, J. Ye, User mobility profile prediction: an adaptive fuzzy inference approach, ACM/Wireless Network 6 (2000) 363–374.
[17] M. Braae, D.A. Rutherford, Fuzzy relations in a control setting, Kybernetes: Int. J. Cybern. Gen. Syst. 7 (1978) 185–188.
[18] L.X. Wang, A Course in Fuzzy Systems and Control, Prentice-Hall, Englewood Cliffs, NJ, 1997.
[19] A. Ghaffari, H.R. Mirkhani, M. Najafi, Stability investigation of a class of fuzzy logic control systems, Proceedings of the IEEE International Conference on Control Applications (2001) 789–793.