SIFS: A Scalable and ISP-Friendly Server Selection Scheme in Cloud Services

Cong Ding¹, Yang Chen¹, Zhuo Chen², Tianyin Xu¹, Long Jin², and Xiaoming Fu¹

¹Institute of Computer Science, University of Goettingen, Germany
²Department of Electronic Engineering, Tsinghua University, Beijing, China
Email: cong@cs.uni-goettingen.de

Abstract. A large number of servers (e.g., CDN servers, datacenter servers) are placed in multiple geographical locations within different network domains. In order to minimize user perceived latency while ensuring high data availability, the server selection problem raises great challenges in modern Internet architecture. This paper presents SIFS, a novel server selection scheme for modern network paradigms exemplified by cloud computing. SIFS has good scalability and ISP-friendliness. SIFS proposes a configurable global performance function that allows ISPs and service providers to leverage the cost (i.e., inter-domain traffic) and the quality of service (i.e., network latency). Results from theoretical analysis and simulations show that, compared with existing server selection approaches, SIFS, with the overall burden increasing linearly with the number of clients, reduces each end user’s measurement cost from $O(N)$ to $O(1)$ for clouds containing $N$ server clusters and decreases about 50% inter-domain traffic.

1 Introduction
In the past few years, the Internet architecture has evolved to the cloud computing era, which provides a powerful infrastructure for hosting online services such as web searching, e-mailing, instant messaging, online social networking, and online gaming. In order to serve the huge and still increasing number of users distributed all over the world, the providers deploy these online services on servers in multiple geo-locations, for example, by reserving content distribution networks (CDNs), or by reserving instances from different datacenters. In this way, the service providers minimize the users’ perceived latency while increasing reliability in case of service outages [5]. Given a fixed set of servers, the service providers are facing the challenge of server selection problem, i.e., how to select server clusters, which means a cluster of co-located servers. Firstly, the selection should improve users’ quality of experience without causing overload on some part of these servers. Secondly, the economic profit of Internet service provider (ISP) is critical, since these globally distributed servers are located within different ISP domains and the inter-domain traffic is expensive.

There are several existing ways for server selection. One straightforward method is completely granting the choice rights to the users. Intuitively, a user
will choose its “closest” server for obtaining the lowest access latency. However, such selection lacks the consideration of the servers’ workload at all, which may lead to outage of some over-capacity servers. Nevertheless, this weakness can hardly be fixed by placing servers according to a certain distribution because users’ online/offline behaviors are highly dynamic and unpredictable. On the other hand, the method imposes high burdens on user clients, especially mobile devices with constrained energy and bandwidth. For example, there are more than 1,158 server clusters in Akamai in 2008 [7], and also thousands of datacenters all over the world [19]. These numbers are still increasing. For a particular user, it takes a long time to finish the measurement. Based on our test by Synaptic Package Manager on Ubuntu version 10.04 LTS using a desktop computer located in University of Goettingen, it takes about 45 seconds for selecting the closest Ubuntu mirror, which consists of 348 servers. Another method is letting service providers or public cloud providers handle the server selection themselves. For example, Akamai uses a centralized hierarchical stable-marriage algorithm for pairing clients with its CDN servers [12]. The centralized architecture causes an extremely large overhead, adds additional delay, and makes the system less responsive to sudden changes in client request rates (i.e., flash crowds), so it is unsuitable to act as an outsourcing system to provide services for all the commercial clouds [15]. However, it puts an unnecessary burden on service providers than outsourcing to a third organization because even the simplest approach requires a whole distributed mapping system including distributed domain name servers (DNSs) and mapping nodes [15]. To solve the problems stated above, some outsourcing mapping systems (e.g., DONAR [15]) made up of dedicated nodes are newly proposed. For example, DONAR considers both proximity and server load in their selection policy. However, such selection scheme is still far from perfect. It cannot scale to larger amount of server locations nor consider the ISP operational cost due to the inter-domain traffic [2, 16].

In this paper, we propose a scalable and ISP-friendly server selection scheme (SIFS) to meet the above challenges. In SIFS, our goals include: (1) Scaling: The SIFS system has to be scalable with the rapid-growing of the service scale due to the huge user scale and wide user distribution. (2) ISP Friendliness: In SIFS, we take the ISP’s economic profit into serious consideration. We aim to reduce the inter-domain traffic while still let users to access their nearby servers. Client nodes should have high probabilities to find the closest server cluster in the same ISP. (3) Decentralization: Centralized systems introduce a single point of failure, as well as an attractive target for attackers. As a result, a decentralized system for server selection is desired. SIFS builds a bridge between end users and service providers for better user-server mapping. By considering different behaviors of the stable servers and dynamic users, we enhance the existing network coordinate (NC) techniques for an accurate latency prediction. Moreover, SIFS proposes a novel global performance optimization for the server selection, by considering the ISP-friendliness. In summary, the paper makes three contributions:

1. We propose a server selection middleware for the cloud services named SIFS, which performs efficient user-server mapping in a decentralized way while
scaling up to wider server distribution, involving several important issues such as proximity, server workload, and ISP-friendliness.

2. We design a distance estimation component of SIFS, which takes the advantage of two different NC techniques (i.e., distributed NC and landmark-based NC) for positioning the user clients with high resilience to churn.

3. We evaluate SIFS based on the real-world Internet traces collected by the Meridian project [1], which reveals that SIFS matches our design goals by satisfying all involved entities (i.e., service providers, ISPs, and end users).

We elaborate SIFS’s system design in Section 2, including the mapping system and the distance estimation system. Section 3 provides the performance evaluation of SIFS when it is applied in the most popular network topology dataset. Section 4 concludes the paper and envisions the future work.

2 SIFS: System Design

There are three kinds of entities in traditional cloud computing service architecture: resource providers, service providers, end users. Resource providers own the cloud computing infrastructure and lease the resources as instances to service providers. Service providers use the resources reserved from the resource providers to serve end users. Traditionally, there are two solutions for the server selection problem: granting the choice rights to the users, and each service provider owning a server selection system. As shown in Fig. 1, SIFS builds a middleware to provide outsourcing selection service, aiming at improving network performance, balancing server workload, and reducing inter-domain traffic.

There are two main components in the SIFS system, mapping system (MS) and distance estimation system (DES). MS allows service providers to choose different selection policies. MS obtains server workload information, network topology, and latencies between servers and clients from three databases respectively, to make server selection decisions based on the algorithm we will describe later. The server workload information is updated by the servers, the network topology database is updated from the RouteViews database [17] (a tool/database for Internet operators to obtain real-time information about the global routing system from the perspectives of several different backbones and locations around the Internet) periodically, and the latencies are updated by DES. DES measures distances between clients and a fixed number of servers which are chosen randomly, using measurement tools such as King [6], a tool to estimate round-trip time (RTT) between any two arbitrary hosts (to which we have no access) in the Internet. The estimated latencies between clients and other servers are calculated based on these measured data and stored to the latency storage database.

2.1 Mapping System

Mapping Policy. We propose to use a global performance function to minimize the network cost, and introduce inter-domain traffic penalty coefficient to balance the reduction of inter-domain traffic and the minimum of user-server latency. In the global performance function, there are two parameters that should
be considered, the inter-domain traffic coefficient and the capacity of each server cluster. The service providers update the inter-domain penalty coefficient directly with any mapping node, to determine the degree of pairing a client with a server in the same domain. Service providers also update the capacity information of each server cluster to the MS periodically. MS obtains ISP information, current workloads of all the server clusters, and network distances between servers and clients, from ISP info database, workload info database, and latency storage database, respectively. In this case, MS solves the global performance optimization problem using the parameters obtained through the policies above, and thus pairs each client to the expected server cluster, as shown in Fig. 2.

**Server Selection Algorithm.** Three important metrics should be considered in the global server selection problem: (1) network performance, (2) server workload balance, and (3) inter-domain traffic. Large latencies between clients and servers cause poor user experiences and network performances, while imbalanced workload of servers may cause a large overhead for specific servers, increasing the risk of server breaking down. The inter-domain traffic produces unnecessary ISP operational cost. Our goal is to minimize the network cost, balance client requests across servers, and reduce inter-domain traffic. However, improving one of these components typically comes at the expense of the others. Thus, we allow our customers (i.e., service providers) configure the parameters to satisfy their willingness for the trade-off among these three factors.

An objective function is desired for seeking an optimal user-server mapping. Subject to the pre-configured load balancing requirement, we try to minimize the latency between every user and the selected server. Furthermore, we introduce inter-domain traffic penalty coefficient, to reflect the ISPs’ economic profit in the objective function. The following global performance optimization problem describes the goals stated above:

$$\begin{align*}
\text{minimize} & \quad \sum_{c \in C} \sum_{i \in I} R_{ci} \cdot cost(c, i) \\
\text{subject to} & \quad B \cdot P_i \leq B_i, \forall i.
\end{align*}$$

where

$$cost(c, i) = \begin{cases} 
D(c, i) & \text{if } c \text{ and } i \text{ belong to the same ISP} \\
\text{penalty}(D(c, i)) & \text{if } c \text{ and } i \text{ belong to different ISPs}
\end{cases}$$
where $D(c, i)$ denotes the latency from client $c$ to server $i$, and $\text{penalty}(D(c, i))$ is the penalty function for inter-domain traffic. Generally we use $\text{penalty}(D(c, i)) = k \cdot D(c, i)$ [14], where $k$ is inter-domain traffic penalty coefficient. The inter-domain traffic penalty coefficient $k = 1$ means the service provider does not distinguish inter- and intra-domain traffic, and $k = +\infty$ denotes the service provider forbid inter-domain traffic by setting inter-domain distance to an infinite value. $C$ and $\mathcal{I}$ are the set of clients and servers. Moreover, the proportion of traffic load $R_{ci}$ that is mapped to server $i$ from client $c$ satisfies $\sum_{i\in \mathcal{I}} R_{ci} = 1$ and $R_{ci} \geq 0$ for any $c$. $B$ is the total amount of traffic, a constant parameter that can be calculated by summing the traffic observed by all the servers. $B_i$ is the capacity of server $i$, and $P_i$ is the proportion of requests directed to server $i$, so that $P_i = \sum_{c\in C} R_{ci} / \sum_{c\in C} \sum_{i\in \mathcal{I}} R_{ci} = \sum_{c\in C} R_{ci} / \# \text{of clients}$.

There are several algorithms and tools to solve this linear programming problem [3], e.g., the simplex algorithm and the criss-cross algorithm. Moreover, the algorithms can be decentralized by enabling each mapping node to perform a smaller-scale local optimization system based on its own view of clients and the aggregated global information collected from other mapping nodes [15].

### 2.2 Distance Estimation System

**DES in a Nutshell.** We propose to use NC techniques as the basic infrastructure for network distance estimation. However, several technical challenges arise in building effective and scalable NC systems in cloud server selection scenario. First, the high dynamics (e.g., mobility, unstable links) of “thin” client devices such as smartphones leads to high churn rate, which greatly deteriorates the performance of traditional P2P-based NC systems. Second, “thin” clients are loosely connected to each other due to their dynamics as well as the constraints on bandwidth and radio coverage. One client cannot have stable neighbor nodes as reference points. As a result, current NC systems with flat structures [4,9,11,13] are no longer applicable for server selection in cloud services.

DES’s design leverages the two features of server selection in cloud computing: (1) Cloud nodes within clouds are highly stable and available [8]; (2) Only the distance estimation between clients and servers are required, while inter-client distances are less important. Fig. 3 demonstrates the system architecture of DES. DES is a two-layer distance estimation system which includes two different kinds of NC systems: the intra-cloud NC system for positioning cloud nodes and the client-cloud NC system for orienting client nodes. For the intra-cloud NC, stable cloud nodes are self-coordinated using decentralized NC systems (e.g., Vivaldi [4], DMF [9]) to obtain accurate servers’ coordinates for
satisfying the large scale of cloud nodes. Further, these cloud nodes can serve as the landmarks (i.e., reference points) to orient the NC estimation of the clients. Since the clients are not involved in the NC calculation of the cloud servers, the overall estimation accuracy of DES will not be impacted by the high churn of client nodes. Consequently, network distances between clients and servers are calculated based on these NCs. In this way, DES reduces each user’s measurement cost from $O(N)$ to $O(1)$ with satisfying estimation accuracy, where $N$ denotes the number of servers within the cloud.

**Intra-Cloud NC System.** DES proposes to employ the decentralized NC system (DNCS) to determine the coordinates of cloud nodes for DNCS’s high estimation accuracy. Since the cloud servers are stable [8], there will be few node churn to impact the overall estimation accuracy. In DES, we choose DMF [9] as the intra-cloud NC system for the following two considerations: (1) DMF utilizes a matrix factorization model, which gets rid of the triangle inequality violation of the Euclidean space and achieves higher accuracy. (2) The regularization component in DMF makes its coordinates stable, which is critical for landmarks. Unstable landmark-based NCs lead to poor estimation performance due to the subsequent high churn of client’s coordinates. Besides creating an NC system which only works for the cloud servers, it is a nature choice to take advantage of the highly available and stable cloud nodes to serve as landmarks for the typical “thin” clients of cloud services. These landmarks enlighten the distance estimation between client nodes and cloud nodes. Different from the existing landmark-based NC systems like GNP [13] and IDES [11], DES gets rid of a centralized algorithm to calculate the NCs of the landmarks so as to achieve high scalability within the cloud.

**Client-Cloud Distance Estimation.** Each DES client randomly selects a subset of the landmarks, and calculates its own NC by referring to the NCs of these landmarks. As in IDES [11], linear least squares are utilized for the NC calculation. Finally, client-cloud distance is estimated as the distance between two coordinates. Landmark-based NC systems have good performance under the high churn of client nodes (not landmarks), which is critical for client-cloud coordinate calculation because of the mobility and dynamics of “thin” client devices (e.g., smartphones) which is common in the cloud-based networks.

**Selective Measurement.** The objective of server selection is to direct client nodes to choose their closest or near-closest cloud nodes within the cloud. However, as reported in [10], the closest neighbor loss of NC systems is significantly large (exemplified by Vivaldi [4]). Thus, it is inaccurate to simply use current NC systems to estimate distances between servers and clients.

We integrate selective measurement (SM) [18] into DES to improve the accuracy for closest server selection, i.e., each client performs another $K$ measurements to the top $K$ closest servers selected by previous NC-based algorithm, and then updates the distance information of all the $K$ links using the real distance obtained by selective measurement. For an $N$-server cloud, we assume that the probability of hitting the actual closest server by selecting the $i$-th closest server based on the estimated distance is $p_i$, where $\sum_{i=1}^{N} p_i = 1$. Then, the probability
of selecting the actual closest server by simply employing NC-based system is
\( p = p_1 \), and the probability by introducing SM is \( p = \sum_{i=1}^{K} p_i \). By introducing
SM with a little extra measurement cost, the probability of selecting the actual
closest server is greatly improved. We show the effect of SM in Section 3.

**Measurement Cost Analysis.** Measurement cost is an important metric in
server selection. High measurement cost makes large bandwidth waste and is impractical for “thin” clients of cloud services. In DES, the measurement cost for each client is \((L + K) \, \text{TM}\) (we define the cost for measuring the distance between
two nodes is 1 time measurement, \( \text{TM} \) for short), where \( L \) denotes the number
of landmarks used in IDES and \( K \) is the number of servers for selective measurement.
Notice that this measurement cost is a constant number and will not
dramatically increase with the growing scale of clients and servers. Comparing
with full measurement with \((N) \, \text{TM}\) cost for each client node, DES’s measurement is much cheaper – DES reduces the cost from \( O(N) \) to \( O(1) \) compared with full measurement. Fig. 3 shows the saved measurement cost compared with full measurement. On the other hand, the measurement cost for each server is \((K) \, \text{TM}\), which is trivial for commercial cloud servers.

3 Performance Evaluation

**Simulation Settings.** InetDim dataset [1] is the only available real-world network latency dataset with autonomous system (AS) topology. It contains all pairs RTTs between 2385 hosts annotated with IP addresses, generated from
the raw King [6] measurements made by the Meridian Project [1]. Based on our
observation, the proportion of ASes containing only 1 node in InetDim dataset
is 70%. In the real Internet, each AS contains a large number of hosts, so we
choose the nodes belonging to the largest 3 ASes from InetDim dataset, which reflects the inter-domain traffic properly. We evaluate the performance of SIFS via simulation using the chosen InetDim dataset, which includes the RTTs between
509 nodes. The impact of the inter-domain traffic penalty is analyzed in a specific way. We compare SIFS with SIFS without selective measurement (denoted as SIFS w/o SM), full measurement (FM) version SIFS (denoted as SIFS w/ FM), and Round Robin algorithm, to show the performance improved by selective measurement, and latency estimation. Note that when inter-domain traffic penalty coefficient \( k = 1 \), SIFS w/ FM is exactly the DONAR system [15]. We also analyze the results when inter-domain traffic penalty coefficient \( k \) changes, and show the performance when the number of servers increases. 100 randomly
chosen nodes serve as servers and the rest as clients; inter-domain traffic penalty coefficient \( k \) is set to 2 generally in our simulation. The other simulation parameters are set according to the values suggested in [9,11,18]: The number of servers for selective measurement is set as \( K = 16 \), while the number of landmarks used
in IDES is set as \( L = 16 \). According to Section 2, each client’s measurement cost
is \( L + K = 32 \, \text{TM} \), i.e., SIFS saves \((1 - 32/100) \times 100\% = 68\% \) measurement cost compared with full measurement (the cost of full measurement is \( N = 100 \, \text{TM} \)).
We run 100 times for each simulation to mitigate the effect of randomness and report the average result values. It takes 3.05 seconds on average for 409 clients
in the 100-server environment in our simulation on matlab 2008b, Windows 7 Professional 64bit, Intel(R) Core(TM)2 Duo CPU E8400 @3.00GHz, 4GB RAM.

**Ranked Order from Closest.** We evaluate SIFS’s performance by comparing inter-domain traffic penalty coefficient $k = 2$ and $k = 1$ (without inter-domain penalty). We measure ranked order from closest (ROFC) [15] as the principal metric to evaluate the performance of the whole selection. ROFC reflects the satisfying degree of user experience. For example, if a client is oriented to the closest server, the selection gives the best user experience. We record the actual ranked order from 1 to 5, i.e., up to the 5th-closest cloud node. With the ranked order, the proportion of selecting the closest server is calculated. Fig. 3 shows that the proportion of selecting closest server by SIFS w/ FM, SIFS, SIFS w/o SM, and Round Robin is 41.68%, 25.79%, 5.77%, 1.03%, respectively, considering inter-domain penalty, and 37.54%, 24.77%, 6.88%, 0.98%, respectively, without considering inter-domain penalty. The proportion of choosing the closest server in SIFS is changed from 25.79% to 24.77%, with only 1.02 percentages decreased. Comparing with the reduced inter-domain traffic (we will show later), it is more than valuable. The reduced proportion from SIFS w/ FM to SIFS is also valuable, comparing with the reduced measurement cost, which is 68% in this simulation, and will be much larger when the number of servers increases.

**Accuracy in Closest Server Selection.** We measure the selection accuracy of SIFS in closest server selection (CSS), i.e., the overhead of all the servers are unlimited. Specially, we randomly select a number of nodes from our dataset as the cloud node, and measure the stretch of using estimation to select the closest server. The stretch is defined as the distance to the closest cloud server cluster selected based on estimation, divided by the distance to the actual closest server cluster [18]. The result of FM is the exact closest server definitely, so the CSS stretch of SIFS w/ FM w/o inter-domain traffic penalty is always 1. Fig. 3 shows the average stretch for choosing different number of servers (same as reported in [18], the average stretch of each curve increases with the number of servers increasing). We see the average stretch value of SIFS is very close to the SIFS w/ FM value and outperforms the other two, and the average stretch value of SIFS is very close to the SIFS w/o inter-domain traffic penalty value. When the number of servers is 50, the average stretch of SIFS is 1.40, much smaller than of SIFS w/o SM (47.01) and round robin (22.16). SIFS’s stretch for 140 servers is only 3.20, much smaller than SIFS w/o SM (85.86) and round robin (26.83). The average stretch of SIFS w/o inter-domain traffic penalty is 1.37,
Impact of Inter-domain Penalty Coefficient. We measure the proportion of choosing closest server, inter-domain traffic, and median latency, when inter-domain traffic penalty coefficient varies. With a randomly chosen 100 nodes serving as cloud servers, we change inter-domain traffic penalty coefficient from 1 to 6 with the step of 0.5. Inter-domain traffic penalty coefficient $k = 1$ means the system does not consider inter-domain traffic penalty, and a large value means a user strongly prefer to choose a server in the same ISP. Fig. 6(a) shows the median latency is not influenced by the value of inter-domain traffic penalty coefficient. Fig. 6(b) shows that the proportion of choosing closest server decreases a little when the inter-domain traffic penalty coefficient increases. From Fig. 6(c), we find that the inter-domain traffic decreases with the value of inter-domain traffic penalty coefficient increasing, and the absolute slope is decreasing. In real application, the penalty coefficient $k$ should be set based on the balance between user-server proximity and inter-domain traffic cost, because different applications’ sensitiveness of proximity and inter-domain traffic cost varies a lot. With the large amount of saved measurement cost and reduced inter-domain traffic, SIFS performs very close to SIFS w/ FM, and outperforms DONAR (SIFS w/ FM with inter-domain traffic penalty coefficient $k = 1$).

Scalability of Mounting Servers. We measure the proportion of choosing closest server, inter-domain traffic, and median latency, when the number of servers increases. Same as the paragraphs above, we randomly select a number of nodes from our dataset as the cloud node, with inter-domain traffic penalty coefficient $k = 2$. Fig. 7(a) shows the median latency decreases with the increasing number of server in a fixed number of ISPs, which means deploying more servers improves the user experiences. To the same, the proportion of choosing closest server and inter-domain traffic decreases when the number of servers in-
creases, in Fig. 7(b) and Fig. 7(c). When the number of servers increases, the measurement overhead of each client is the same, which shows the scalability in terms of the increasing number of servers.

4 Conclusion and Future Work

In this paper, we propose SIFS, a novel server selection scheme for cloud-based services. SIFS is scalable to deal with the explosively-increasing numbers of clouds as well as user clients based on its DES component. DES takes the advantages of two different kinds of NC techniques (i.e., the distributed NC and landmark-based NC) to position both the clouds and users, achieving the measurement cost reduction from $O(N)$ to $O(1)$ for clouds with $N$ server clusters. In addition, SIFS is ISP-friendly that effectively reduces inter-domain traffic leading to low ISP operational costs based on its MS component which makes a balance between user quality of experience (the closest server selection) and the operation cost (inter-domain traffic), in the limitation of servers’ capacity.

Our future work includes the following two aspects. First, we expect to define a complete application program interface (API) for fast and convenient configurations. Second, we would like to collect larger network distance datasets with ISP information to evaluate the performance in large scale network environment and the feasibility of serving many service providers at the same time.

References